Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Documentation

The following documentation items are available:

- PCRShell: using PCRaster
- PCRaster visualisation commands:

Items available	Display	Aguila	Timeplot
manual	Yes	Yes	Yes
screenshots	Yes	Yes	No
faq	No	Yes	No
known problems	Yes	Yes	No
bugs	Yes	No	No

- Gstat manual: you can find this at the Gstat homepage.
- All other PCRaster commands containing
    1. list of pcrcalc functions
    2. PCRaster applications
- Updates and additions to above documentation

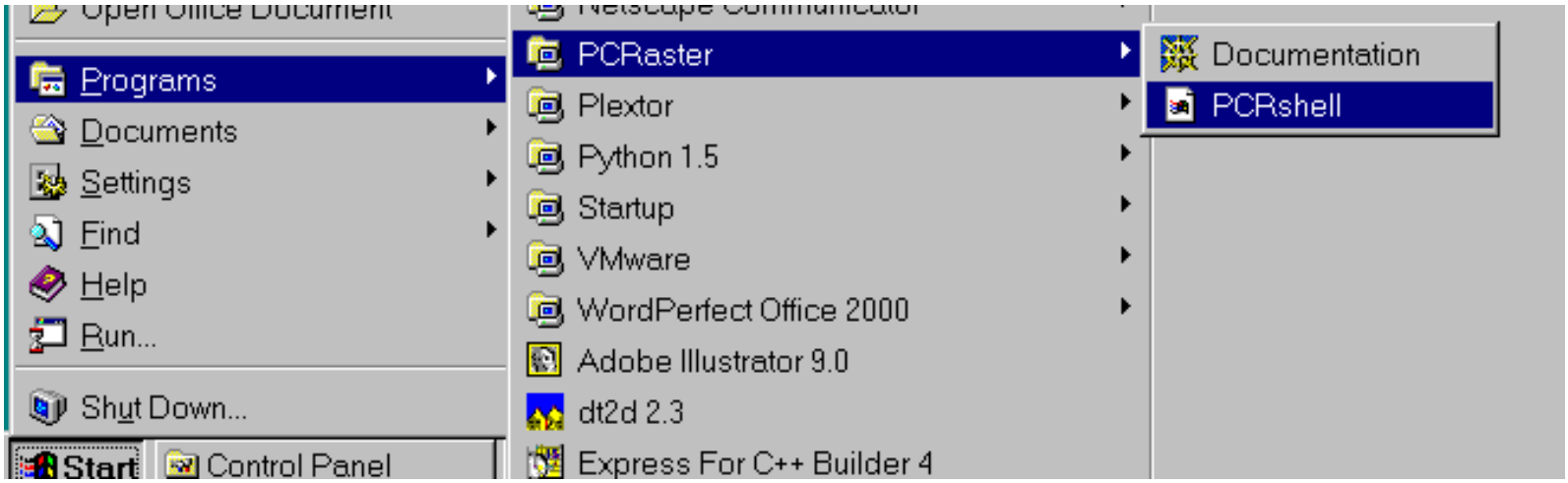
Clickpath: Documentation Versions: page: 20 November 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences Utrecht University
The Netherlands

Universiteit Utrecht

PCRShell

On Microsoft Windows operating systems, PCRaster commands are given from the command line shell, also known as the MS-DOS prompt. If PCRaster is correctly installed on your computer will the MS-DOS prompt of PCRaster under the Start menu, in the Programs/PCRaster folder you find PCRShell, As illustrated in the image below. PCRShell allows to recall and change previously typed commands by pressing the up and down keys of the keyboard. It also always start in the directory (or folder) where you finished working with PCRaster the last time. When PCRShell is started the first time after installation, it will start in the directory named workspace. From there you can go to the (sub-)directory where the data for the Map Algebra course is installed.



Clickpath: PCRShell Versions: page: 19 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Display Manual

Next

Display Manual

Software to visualise rasters

Kor de Jong

Utrecht University
Faculty of Geographical Sciences
Department of Physical Geography

Heidelberglaan 2
80115
3508 TC
Utrecht
Netherlands
<k.dejong at geo.uu.nl>

Table of Contents

1. License
2. Introduction
3. Usage
    - Syntax
    - Options
    - Filenames
4. Interface
5. Changing the looks
A. Colourpalette file format
B. Menu-bar
C. Shortcuts

List of Figures

4.1. Display interface

List of Tables

3.1. Command line options

B.1. File menu options

B.2. Edit menu options

B.3. View menu options

B.4. Help menu options

C.1. Shortcuts to Display functions

List of Examples

A.1. Colourpalette for soil map with 5 classes.

Next

Chapter 1. License

Chapter 1. License

Prev Next

Chapter 1. License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should receive a copy of the GNU General Public License along with the programs; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Prev Up Next
Display Manual Home Chapter 2. Introduction

Chapter 2. Introduction

Prev Next

Chapter 2. Introduction

This is the user manual of the Display application. The current version of Display can be used to visualize PCRaster map layers.

The software is available for MS-Windows 95/98/NT and Linux. I assume that you have a working copy of Display installed.

In Chapter Chapter 3, *Usage* of this manual I will show you how to start Display. There are a few command-line options which I will describe.

Display visualizes map layers based on the type of information in the layer. For example, a DEM is visualized differently from a soil map. In Chapter 3 I will show you how you can use the interface to change the default visualization of a map layer.

Prev Up Next
Chapter 1. License Home Chapter 3. Usage

Chapter 3. Usage

Prev Next

Chapter 3. Usage

Syntax

The Display command has the folowing syntax:

```
$ display [-p filename] [-q filename] [-r interval] [-w caption] [filename(s)]
```

Prev | Up | Next
Chapter 2. Introduction | Home | Options

Options

Table 3.1. Command line options

Option	Description
-p f	Read colour palette from file f. All map layers will be drawn using this palette. See Appendix A, *Colourpalette file format* for a description of the file format for colour palettes.
-q f	Create lock file f. If f does not exist it is created. When Display exists f is deleted again. This can be useful if Display is started by another application which wants to be able to check if Display is still running.
-r #	Set animation interval to # miliseconds.
-w s	Set the captions of all windows to s.

Filenames

A filename can be the name of a static map layer or the name of a dynamic stack data. A static map layer is a layer of data which doesn't change in time. A dynamic map layer is a layer of data which changes in time. The name of a static map layer is just the filename of the csf-file which contains the data. The name of a dynamic map layer is the filename of the csf-file which contains the data of the first time step concatenated by a '+' and the number of the last time step.

To visualize a static soil map, issue the following command:

```
$ display soil.csf
```

To visualize a dynamic soil map, issue the following command:

```
$ display soil0000.001+100
```

If you want more than one map layer in one window, separate the filenames of the layers with ' + '. Note that the spaces around the '+' are relevant!

To visualize a dynamic dtm with an ldd on top of it, issue the following command:

```
$ display dem00000.001+120 + ldd.csf
```

Appendix A. Colourpalette file format

A colourpalette file is an ASCII file with at least three columns. The first three columns contain the rgb values / tupples of the colours in the palette. Rgb values range from 0 to 255.

Example A.1. Colourpalette for soil map with 5 classes.

```
163 83  5
109 96  79
210 143 64
2   118 117
100 1   125
```

Chapter 4. Interface

Figure 4.1, Display interface shows the Display interface if no map layers are loaded.

See Appendix B, *Menu-bar* for a description of all menu items of the menu bar. The tool bar contains shortcut buttons to most of the menu items of the menu bar. Finaly, most menu items of the menu bar have shortcut keys attached to them. See Appendix C, *Shortcuts* for a list of shortcut keys.

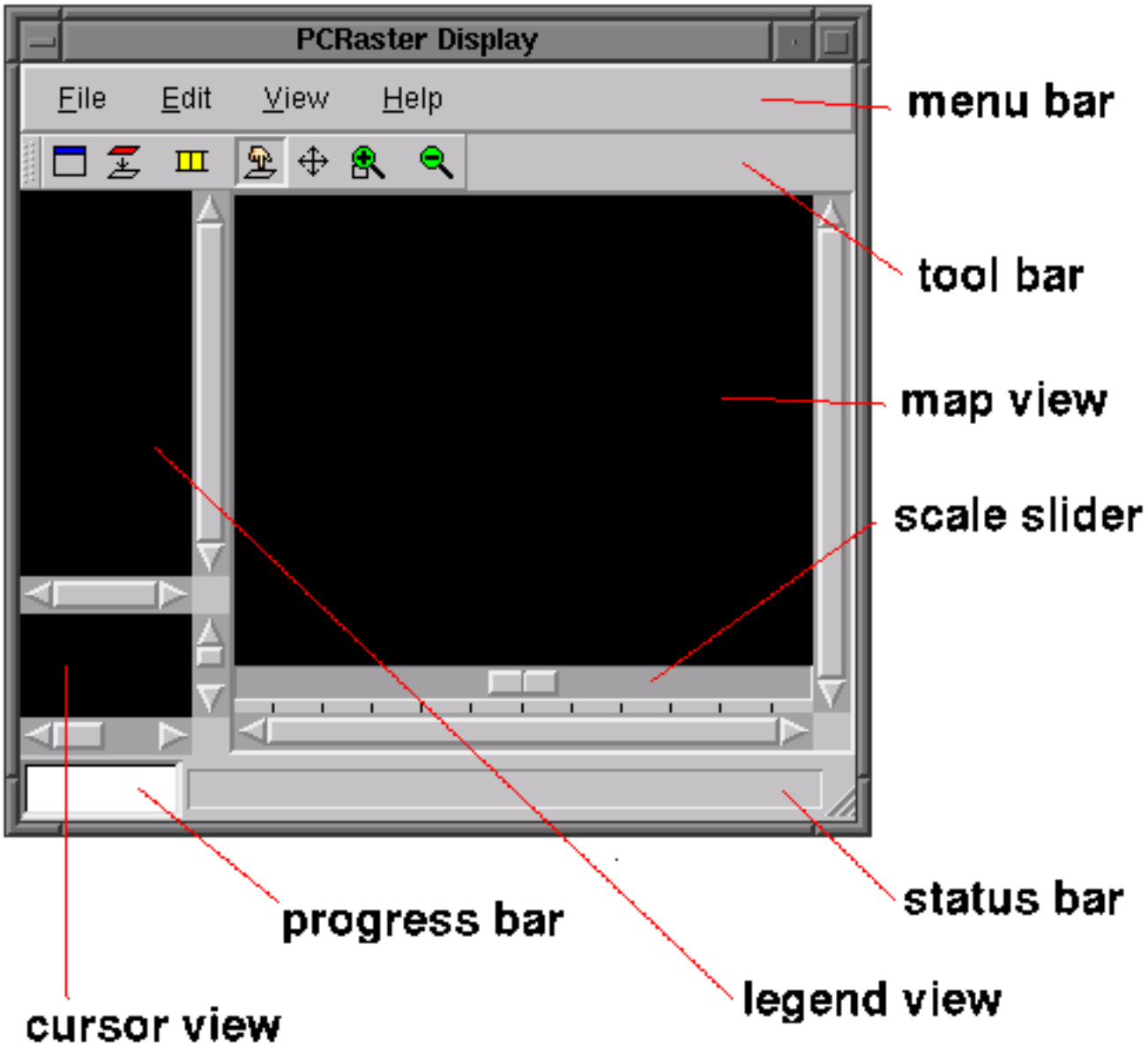


Figure 4.1. The Display interface.

The map view is the area of the interface where the map layers are drawn.

The legend view is the area of the interface where the legend of the map layers are drawn.

The cursor view is the area of the interface where information about the current cursor position will be drawn.

Prev | Up | Next
Filenames | Home | Chapter 5. Changing the looks

Chapter 5. Changing the looks

Chapter 5. Changing the looks

Based on the type of information of a map layer Display decides how to visualize it. For example, a layer with continuous values is drawn using other colours than a layer with classified values.

The interface enables you to change the properties of the map drawer of a map layer. If you double-click the left mouse button on a legend of a map layer, than a properties dialog for the drawer of that layer will appear. The same can be achieved by clicking the right mouse button on a legend and selecting the Properties... menu item from the appearing popup menu.

Appendix B. Menu-bar

Table B.1. File menu options

Menu item	Description
New display window	Creates and shows a new, empty display window.
Open layer...	Opens, reads and shows a new map layer. The new layer is put on top of the current map layers in the view.
Save layer	Saves the current map layer and colour palette. This option is only available if there's one map layer in the view.
Save layer as...	Saves the current map layer and colour palette in a new file. This option is only available if there's one map layer in the view.
Close	Closes the map view. If there're no more map views visible than the Display application will quit.
Exit	Closes all map views and exists the Display application.

Table B.2. Edit menu options

Menu item	Description
Preferences...	Shows a dialog with which some 'view-wide' settings can be manipulated.

Table B.3. View menu options

Menu item	Description
Toolbar	Shows or hides the toolbar.
Status bar	Shows or hides the status bar.
Animate	Starts the animation of dynamic data. Nothing will happen if no dynamic data is loaded.
Point	Sets the cursor in point mode. This enables you to change the cursor of the map by clicking on the map.
Pan	Sets the cursor in pan mode. In pan mode you can pan the map by clicking and dragging it with the left mouse button pressed down.
Zoom...	Shows the zoom menu. Here you can choose between 'Zoom area' and 'Zoom all'. 'Zoom area' sets the cursor in zoom mode. To zoom an area of the map, select a corner with the left mouse button pressed and create a zoom rectangle by dragging the mouse. 'Zoom all' zooms the layers to their bounding rectangle.

Table B.4. Help menu options

Menu item	Description
About...	Shows some information about the Display application.

Prev Up Next
Appendix A. Colourpalette file format Home Appendix C. Shortcuts

Appendix C. Shortcuts

Prev

Appendix C. Shortcuts

Table C.1. Shortcuts to Display functions

Function	MS-Windows	Linux
new Display window	Ctrl-N	Alt-N
open layer	Ctrl-O	Alt-O
save layer	Ctrl-S	Alt-S
close	Ctrl-W	Alt-W
exit	Ctrl-Q	Alt-Q
animate	Ctrl-A	Alt-A
point	Esc	Esc
pan	Ctrl-M	Alt-M
zoom area	Ctrl-I	Alt-I
zoom all	Ctrl-Z	Alt-Z

Prev Up
Appendix B. Menu-bar Home

User manual of Aguila

Next

User manual of Aguila

Kor de Jong

Utrecht University
Faculty of Geographical Sciences
Department of Physical Geography

Heidelberglaan 2
80115
3508 TC
Utrecht
Netherlands
<k.dejong at geo.uu.nl>

Table of Contents

1. Concepts
2. Installation
3. Starting Aguila
4. Using Aguila
    - Control center
    - Map2D
        - Animation
        - Controls
    - Map3D
        - Animation
        - Controls
        - Camera's
        - Shortcuts
    - Visualization groups

A. Shortcuts

B. Static versus dynamic linkage

Note for Linux users

C. Qt GUI Toolkit

D. OpenGL Graphics Library

Glossary

List of Figures

1.1. A subject, consisting of a stack, a timeseries and a cursor position.

1.2. Three observers: a 3D map, a timeplot and a 2D map visualization.

1.3. A group: three observers observing one subject.

List of Tables

3.1. Command line arguments

4.1. Controls for changing the scale and position of the map.

4.2. Controls for changing the position and orientation of the camera.

4.3. Controls for changing the orientation of the scene.

4.4. Camera's.

4.5. Shortcuts.

A.1. Shortcuts to Aguila functions

Next

Chapter 1. Concepts

Chapter 1. Concepts

Prev Next

Chapter 1. Concepts

It is important to understand a few concepts on which the software is based. The most important concepts are the notion of subjects, observers and their relationships within visualization groups.

A subject is a combination of data and state. Data can be a stack of raster layers containing elevation values or a timeseries with soil classes for a certain location or both or any other (combination of) data which the software is willing to read. State is the current cursor position. Apart from data, a subject also keeps track of the current cursor position. A cursor position is a certain position in space and time which is selected by the user, for example by clicking the mouse in a 2D map visualization. See also Figure 1.1, A subject, consisting of a stack, a timeseries and a cursor position. .

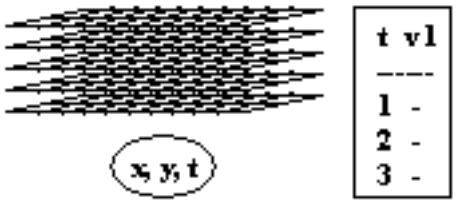


Figure 1.1. A subject, consisting of a stack, a timeseries and a cursor position.

An observer is a visualization of some data. Examples of observers are: a 3D visualization of a raster layer with elevation values or a timeplot or a 2D visualization of a raster layer with soil classes. See also Figure 1.2, Three observers: a 3D map, a timeplot and a 2D map visualization. .

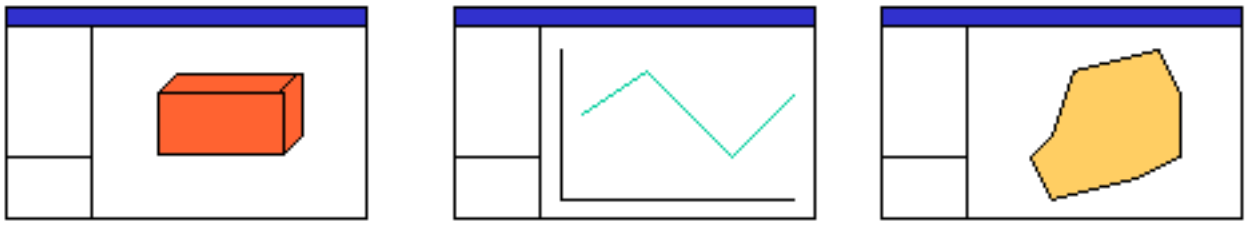


Figure 1.2. Three observers: a 3D map, a timeplot and a 2D map visualization.

Furthermore, a subject can be observed by one or more observers: data can be visualized by one or more visualizations. And an observer is always connected to exactly one subject: a visualization is always connected to a certain piece of data. A subject and its connected observers are called a visualization group or a group in short. See also Figure 1.3, A group: three observers observing one subject.

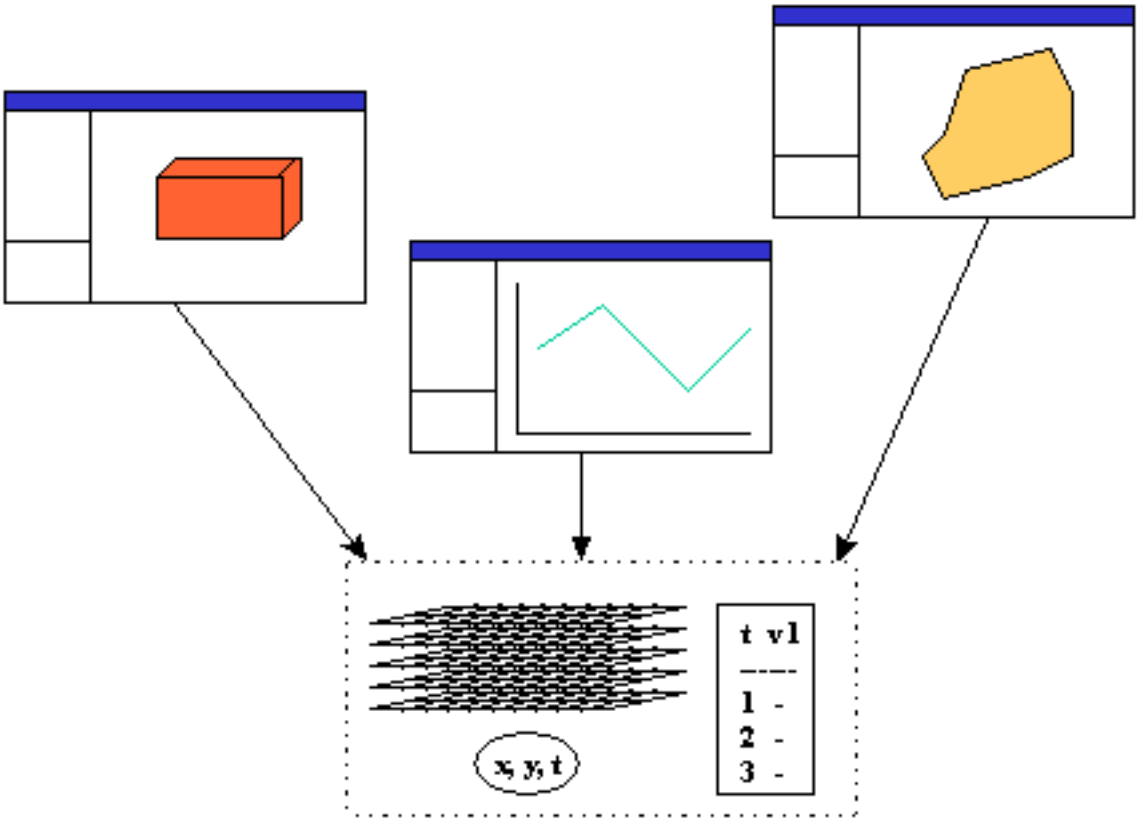


Figure 1.3. A group: three observers observing one subject.

If the current cursor position changes, the subject will notify all its observers that it changed its state. The observers than detect the change and, if needed, update their content. So, if you start an animation in a certain visualization, the time-part of the current cursor position is continuously changed. The subject notifies all observers that the time changed and, if needed, they all update their contents.

Chapter 2. Installation

Prev Next

Chapter 2. Installation

The software is distributed in an archived directory structure. If you unpack the archive, a new directory is created called `aguila` with a version string appended to the name (e.g.: `aguila-000331`). Below it you will find the folowing directories:

`bin`

Contains the executable.

`doc/html`

Contains the manual in html-format.

To be able to use the executable you need to copy it to a location in your search path or adjust your search path variable to include the `bin` directory.

Prev Up Next
Chapter 1. Concepts Home Chapter 3. Starting Aguila

Chapter 3. Starting Aguila

`aguila` [[[-2] | [-3]]] [-q *`lock file`*] [*`stack name`* | *`stack name + stack name`*]...

Table 3.1. Command line arguments

Argument	Description
-2	Start with 2D map visualization(s).
-3	Start with 3D map visualization(s).
-q lock file	Create lock file with name *`lock file`*. If *`lock file`* does not exist it is created. When Aguila exists *`lock file`* is deleted again. This can be useful if Aguila is started by another application which wants to be able to check if Aguila is still running.

Without options, Aguila will try to find default visualizations for the *`stack name`* arguments. You can force a certain kind of visualization by using the -2 or -3 commandline options for 2D or 3D visualizations respectively. If an argument cannot be visualized, it will only be loaded into the control center.

The *`stack name`* argument is the name of a static or a dynamic stack. In case of a static stack you should just provide the filename. The name of a dynamic stack is the filename of the first raster layer you want to visualize concatenated by a '+' and the number of the last time step.

Stack names which are seperated by ' + ' (note the spaces!) are draped on top of each other.

Aguila will try to put as many data and visualizations in the same visualization group as possible. Currently it adds each new stack on the command line to the current visualization group. But if the spatial properties of that stack differ from the properties of the group, than a new group is created and the stack added to that one.

If the *`stack name`* argument is skipped or if Aguila has no default visualization for the loaded data, than Aguila will start with the control center.

For example, to view a static dem in 3D, issue the following command:

```
$ aguila dem.csf
```

or

```
$ aguila -3 dem.csf
```

If you want it in 2D, issue the following command:

```
$ aguila -2 dem.csf
```

To view a dynamic dem in 3D, issue the following command:

```
$ aguila dem00000.001+100
```

To drape a soil map on a dem, issue the following command (both stacks can also be dynamic):

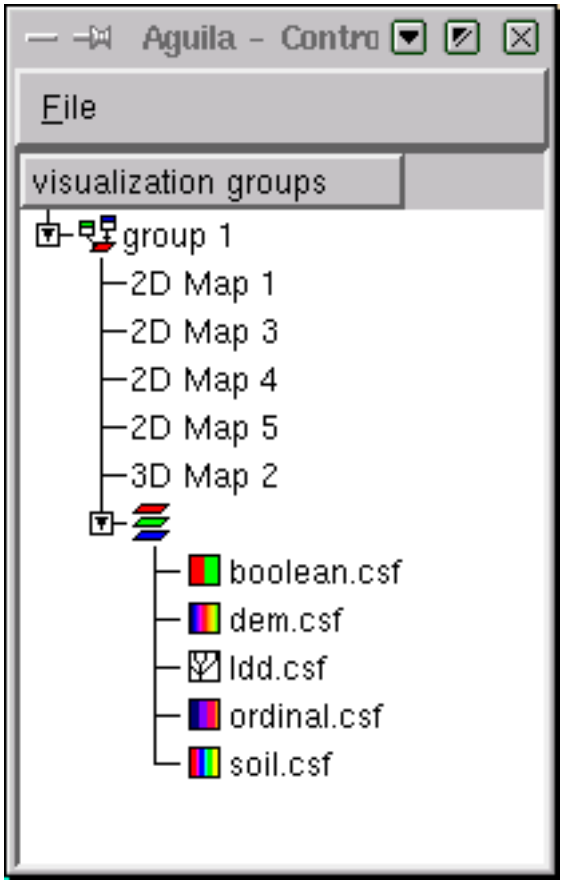
```
$ aguila dem.csf + soil.csf
```

Prev | Up | Next
Chapter 2. Installation | Home | Chapter 4. Using Aguila

Chapter 4. Using Aguila

If you follow the instructions in Chapter 3, *Starting Aguila*, than Aguila will show your data in a default visualization (currently Map2D and Map3D are implemented) or it will show the control center.

Control center



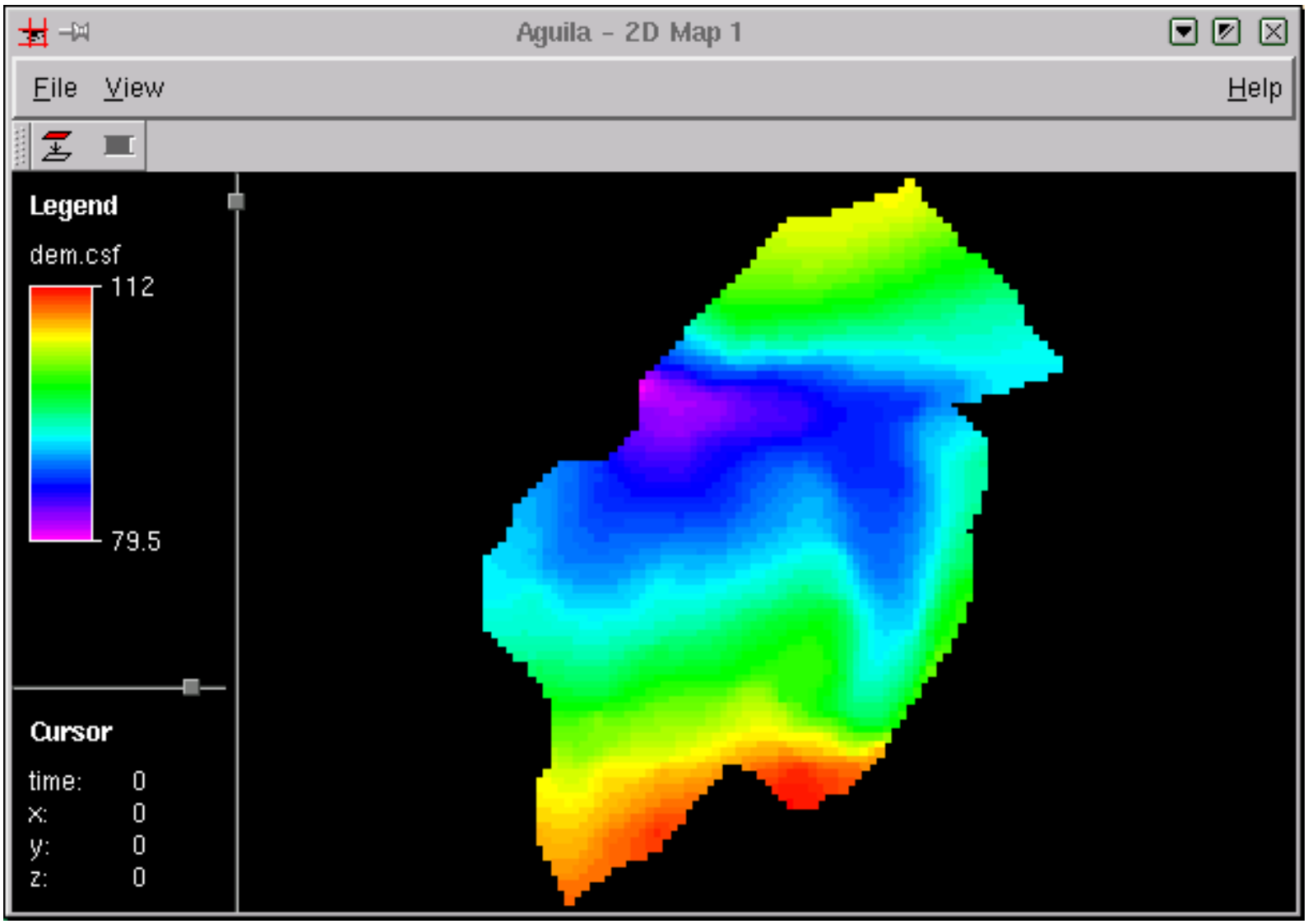
From the control center you can create new visualization groups, close existing groups, add data to the data subject and add visualizations to a group. Click with the right mouse button on the various parts of the interface for a popup menu with available options.

Note

Currently you don't have much control in the control center. More functionality will be added in the future. It is useful now for keeping track of the visualizations and data in visualization groups.

Map2D

The Map2D visualization shows PCRaster stacks as 2D maps.



Example of Map2D visualizing a digital elevation model.

Animation

If you loaded a dynamic stack, than you can start an animation of the stack by selecting the Animate menu item from the View menu, or by pressing the Animate toolbutton ( ) from the toolbar. A dialog will be shown from which you can control the animation. Remember that by animating you change the time part of the current cursor position of the data subject (see Chapter 1, *Concepts*). This will trigger an update of all visualizations observing that data subject. This means that the animation control dialog is global to the data subject and controls all its observers.

Controls

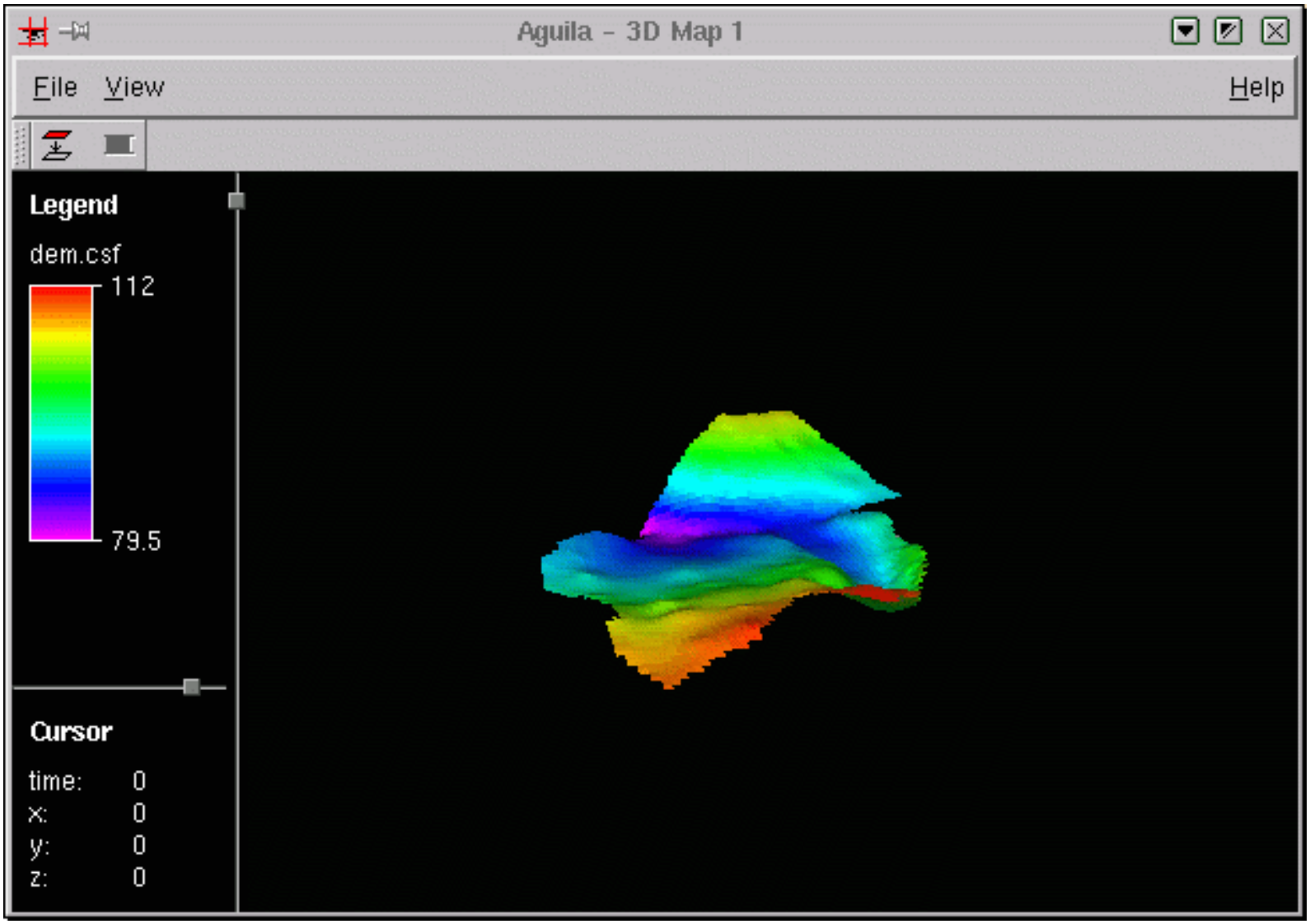
The folowing controls are implemented to change the scale and position of the map:

Table 4.1. Controls for changing the scale and position of the map.

Effect	Control
move map to the right	**h**
move map to the left	**l**
move map to the bottom	**k**
move map to the top	**j**
zoom in	**Ctrl-k**
zoom out	**Ctrl-j**
reset	**r**

Map3D

The Map3D visualization shows a scalar raster as a surface (or sheet) which floats in space. You can change some properties of the 3D map visualization by right clicking your mouse in the map view (the one which shows the sheet) and selecting Properties. The properties dialog for the map view will be shown.



Example of Map3D visualizing a digital elevation model.

Animation

If you loaded a dynamic stack, than you can start an animation of the stack by selecting the Animate menu item from the View menu, or by pressing the Animate toolbutton ( ) from the toolbar. A dialog will be shown from which you can control the animation. Remember that by animating you change the time part of the current cursor position of the data subject (see Chapter 1, *Concepts*). This will trigger an update of all

visualizations observing that data subject. This means that the animation control dialog is global to the data subject and controls all its observers.

Controls

There's a difference between controlling the camera (your head) and the 3D scene. You can control the position and aim of the camera (see Table 4.2, Controls for changing the position and orientation of the camera.). You can only control the orientation of the scene (see Table 4.3, Controls for changing the orientation of the scene.). Together these controls enable you to look at every part of the scene from everywhere.

Table 4.2. Controls for changing the position and orientation of the camera.

Effect	Control
look left	**h** / left mouse drag left / right mouse drag left
look right	**l** / left mouse drag right / right mouse drag right
look up	**k** / right mouse drag forwards
look down	**j** / right mouse drag backwards
roll clockwise	**n**
roll counter-clockwise	**m**
move left	**Shift-h** / left-right mouse drag left
move right	**Shift-l** / left-right mouse drag left
move up	**Shift-k** / left-right mouse drag forwards
move down	**Shift-j** / left-right mouse drag backwards
move forward	**Shift-Up** / **Ctrl-k** / left mouse drag forwards
move backwards	**Shift-Down** / **Ctrl-j** / left mouse drag backwards
reset	**r**

Table 4.3. Controls for changing the orientation of the scene.

Effect	Control
rotate clockwise around z-axis	**Left**
rotate counter-clockwise around z-axis	**Right**
rotate clockwise around x-axis	**Up**
rotate counter-clockwise around x-axis	**Down**
reset	**r**

The camera and the scene have their own coordinate system which can be moved and rotated independently from each other. You should interpret the above controls relative to the direction of the coordinate system of the camera or the scene. For example, moving the camera forward means moving the camera in the direction of the camera. This might be in a direction away from the scene!

Camera's

Apart from the camera whose position and orientation can be changed, there are 5 more camera's in the scene. These static camera's are positioned in such a way that it is possible to see the relative position of the scene and the mobile camera. Table 4.4, Camera's. gives a list of camera's you can choose from (note the correspondence between the shortcuts and the layout of the numeric keypad on your keyboard).

Table 4.4. Camera's.

Camera	Shortcut
Top	**5**
Front	**2**
Left	**4**
Back	**8**
Right	**6**
Mobile	**0**

Shortcuts

Table 4.5. Shortcuts.

Effect	Shortcut
exaggerate heights	**Plus (+)**
understate heights	**Minus (-)**
enlarge quad length	**Shift-q**
smallen quad length	**q**

Visualization groups

As explained in Chapter 1, *Concepts* visualization groups consist of data, a cursor and visualizations. As said there, a change in cursor position will trigger updates of the visualizations if that makes sense. For example, if you load two dynamic stacks into a visualization group and you visualize them with the Map2D and/or Map3D visualizations, then animating one of them will also animate the other. In fact, the animation dialog only changes the cursor position of the group, which in turn, notifies all visualizations of the cursor change.

Synchronizing visualizations by visualization groups enables you to quickly query your datasets for information. But sometimes it is useful to create different visualization groups for the same data sets. For example, if you want to compare data for different time steps with each other, then you must put the data in different visualization groups.

Note

Future releases of the software will contain more 'visualization group - related' functionality. For example, it would be very useful to be able to copy a whole visualization group so you can easily compare data for different time steps with each other. To do that now, you have to create a new group, add visualizations to it and load the data.

Appendix A. Shortcuts

Table A.1. Shortcuts to Aguila functions

Function	MS-Windows 95/98/NT	Linux
open file	**Ctrl-o**	**Alt-o**
close	**Ctrl-w**	**Alt-w**
exit	**Ctrl-q**	**Alt-q**

Appendix B. Static versus dynamic linkage

There're two versions of the software: a statically linked executable and a dynamically linked executable. Most non-trivial applications depend on one or more libraries with re-usable software components. If you're using a MS-Windows operating system: have a look at the `*.dll` files in the `c:\windows\system` directory (or similar) to see what I mean.

If an application is linked dynamically to a library, it means that it searches and reads the library at run-time. If an application is linked statically to a library, it means that (part of) the library is copied into the executable.

The obvious advantage of dynamically linked executables is that they can share the re-usable, common software components they use. This will cut down the number of bytes occupied by applications on your harddisk.

The advantage of statically linked executable is that you don't need to install anything else but the executable to be able to run it.

For all the graphical stuff used by Display it depends on the Qt GUI Toolkit (see Appendix C, *Qt GUI Toolkit*). If you download the statically linked version of Display, the Qt-library is linked into the executable. If you download the dynamically linked version of Display, you have to make sure you install the Qt-library yourself.

Note for Linux users

After you have installed the Qt-library you have to make sure that the dynamic loader can find it. The file `/etc/ld.so.conf` contains a list of directories which the loader searches for shared libraries. If the directory where you installed the qt-library isn't listed in ld.so.conf then the loader won't find it. One solution is to edit `/etc/ld.so.conf` . Another solution is to set the environment variable `LD_LIBRARY_PATH` to contain the directory where the Qt-library is installed.

Since the Qt-library isn't part of the Linux/X core installation I recommend that you install the library in `/usr/local/lib` and add that path *to the end* of `/etc/ld.so.conf`. You have to run the ldconfig command (as root) before any changes in `/etc/ld.so.conf` have any effect. For a critical discussion about this subject, have a look at the homepage of David Barr (www.visi.com/~barr/ldpath.html).

Appendix C. Qt GUI Toolkit

Prev Next

Appendix C. Qt GUI Toolkit

Qt is a graphical user interface (GUI) software toolkit. Qt simplifies the task of writing and maintaining GUI applications. We use it for PCRaster's graphical applications (e.g.: Display and Aguila). Normally you couln't care less about the name of the GUI toolkit used but if you want to download a dynamically linked version of our software (see Appendix B, *Static versus dynamic linkage*), than you need to install Qt yourself.

You can find all information about Qt at the homepage of Troll Tech: www.trolltech.com.

Prev | Up | Next

Appendix B. Static versus dynamic linkage | Home | Appendix D. OpenGL Graphics Library

Appendix D. OpenGL Graphics Library

Prev Next

Appendix D. OpenGL Graphics Library

Aguila uses the OpenGL Graphics Library (www.opengl.org) for the rendering of 3D objects (Map3D). OpenGL is a standard software interface which is implemented on a number of platforms (e.g.: Window 95/98/200/NT, Linux, Macintosh). If it is not already installed on your computer, visit www.opengl.org/applications/installing.html.

It is possible to improve the speed of an application which uses OpenGL by using a 3D accelerator with OpenGL driver. A list of graphics boards which are OpenGL enabled can be found at www.opengl.org/applications/graphic_cards.html. If you're about to buy a new computer, I recommend including such a board.

Prev Up Next
Appendix C. Qt GUI Toolkit Home Glossary

Glossary

O

observer

A combination of data and state.

See Also subject.

S

subject

A visualization of data.

See Also observer.

Timeplot Manual

Next

Timeplot Manual

Software to visualise timeseries

Kor de Jong

Utrecht University
Faculty of Geographical Sciences
Department of Physical Geography

Heidelberglaan 2
80115
3508 TC
Utrecht
Netherlands
<k.dejong at geo.uu.nl>

Table of Contents

1. License

2. Description

3. Usage

4. Interface

Next

Chapter 1. License

Chapter 1. License

Chapter 1. License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should receive a copy of the GNU General Public License along with the programs; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Chapter 2. Description

Prev Next

Chapter 2. Description

The Timeplot command can be used to visualize time series output by PCRaster. Timeplot shows time series as two dimensional line plots. More than one time series can be shown in one plot.

Prev Up Next
Chapter 1. License Home Chapter 3. Usage

Chapter 3. Usage

Chapter 3. Usage

The Timeplot command has the folowing syntax:

```
$ timeplot [filename(s)]
```

If you put a '+', surrounded by spaces, between two filenames, than the time series in both files will be shown in one plot

Chapter 4. Interface

Prev

Chapter 4. Interface

The interface of Timeplot has three areas were information about the time series are shown: the plot view, the legend view and the cursor view.

The plot view shows the time series as a two dimensional line plot. You can change the properties of the line plot by pressing the right mouse button on one of the axes or in the plot area itself, and selecting Properties.

The legend of the time series in the plot view is shown in the legend view. The legend items are grouped by file. You can change the properties of the legend items of a file by pressing the right mouse button on one of the legend items in the group and selecting Properties. By double clicking the left mouse button on a menu item you can change the properties of one menu item.

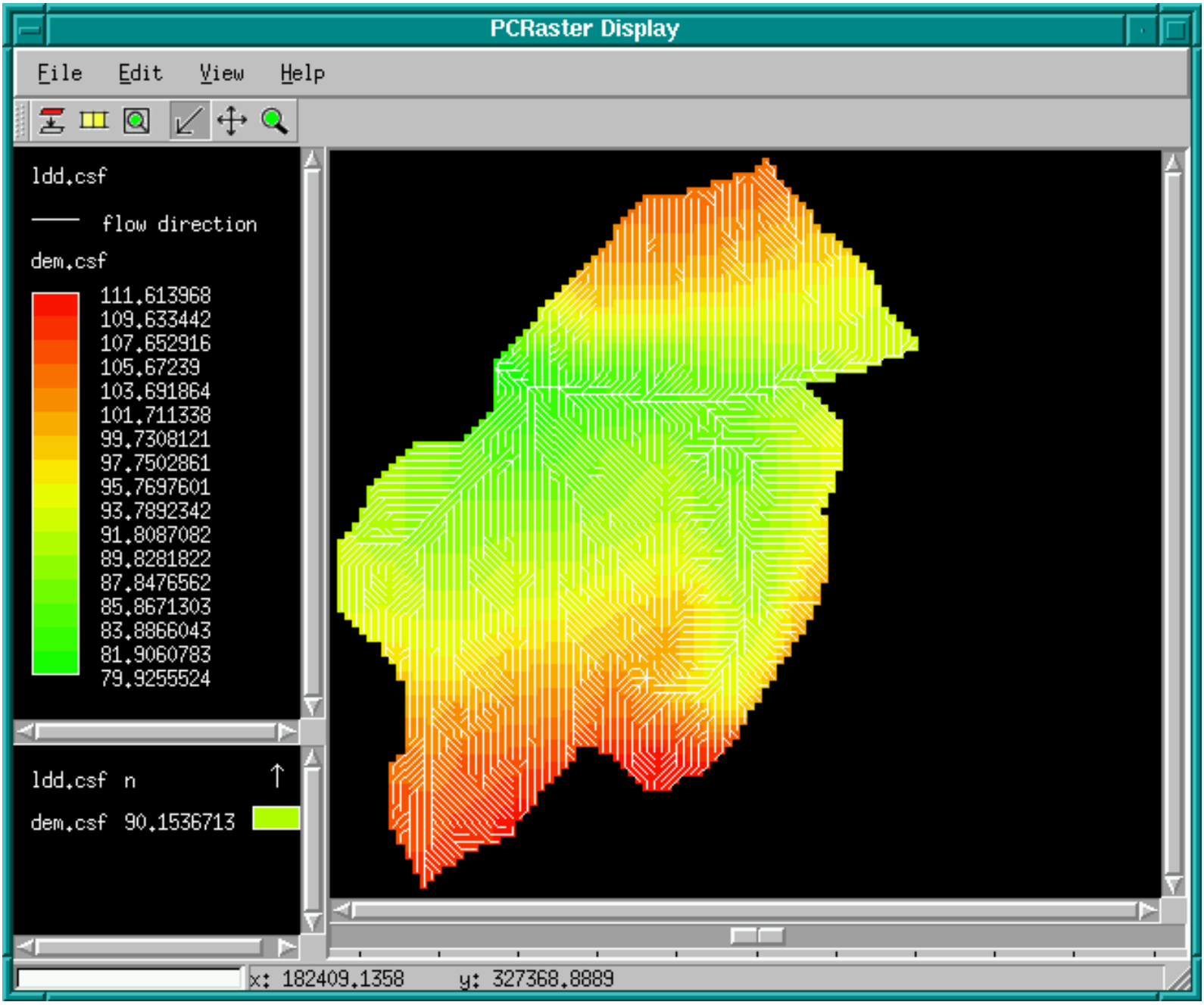
The cursor view shows the current cursor settings: location, time and attribute values. You can change the cursor by pressing the left mouse button in the plot area in the plot view.

Prev Up
Chapter 3. Usage Home

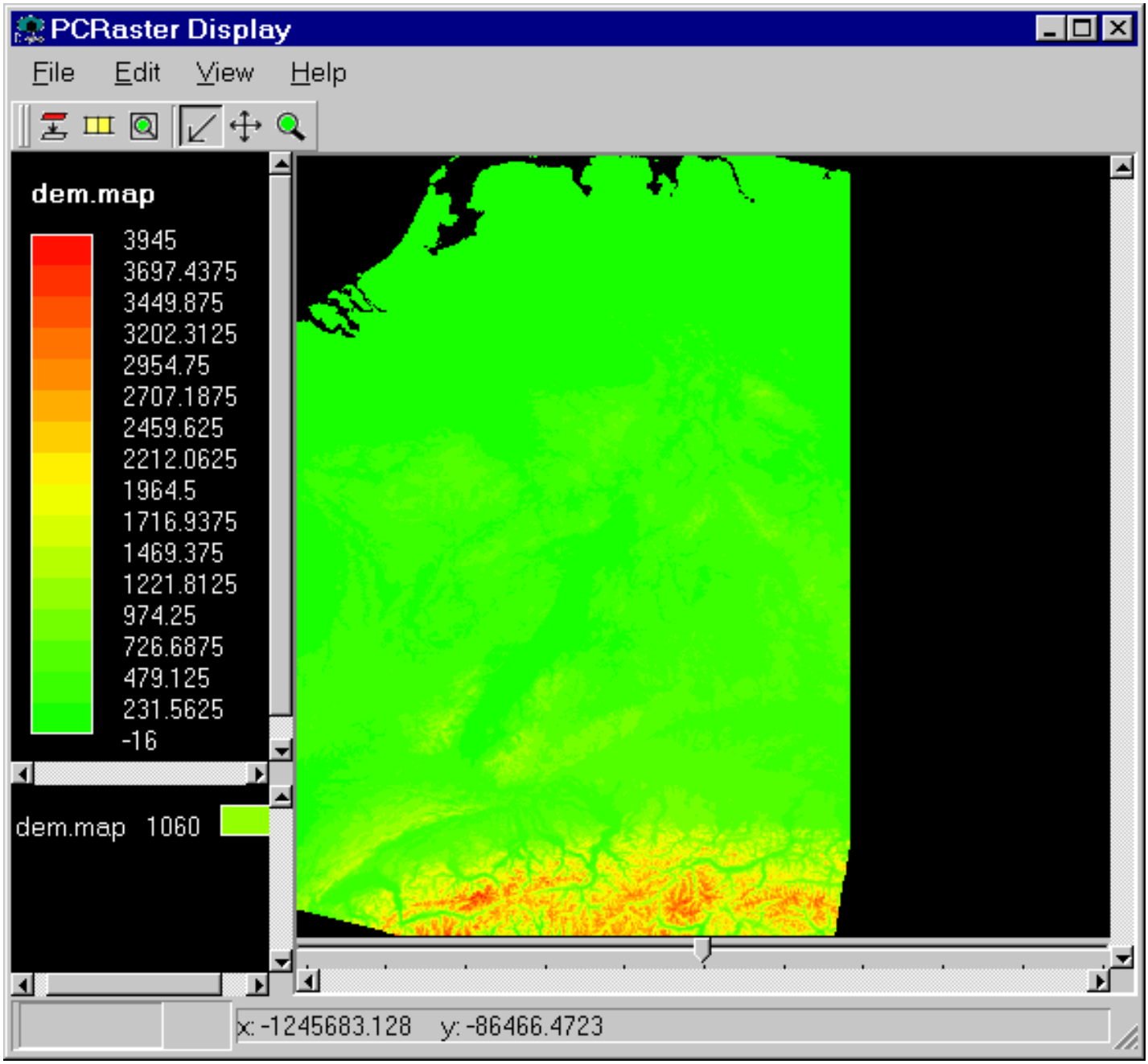
Department of Physical Geography Faculty of Geosciences Utrecht University The Netherlands

Universiteit Utrecht

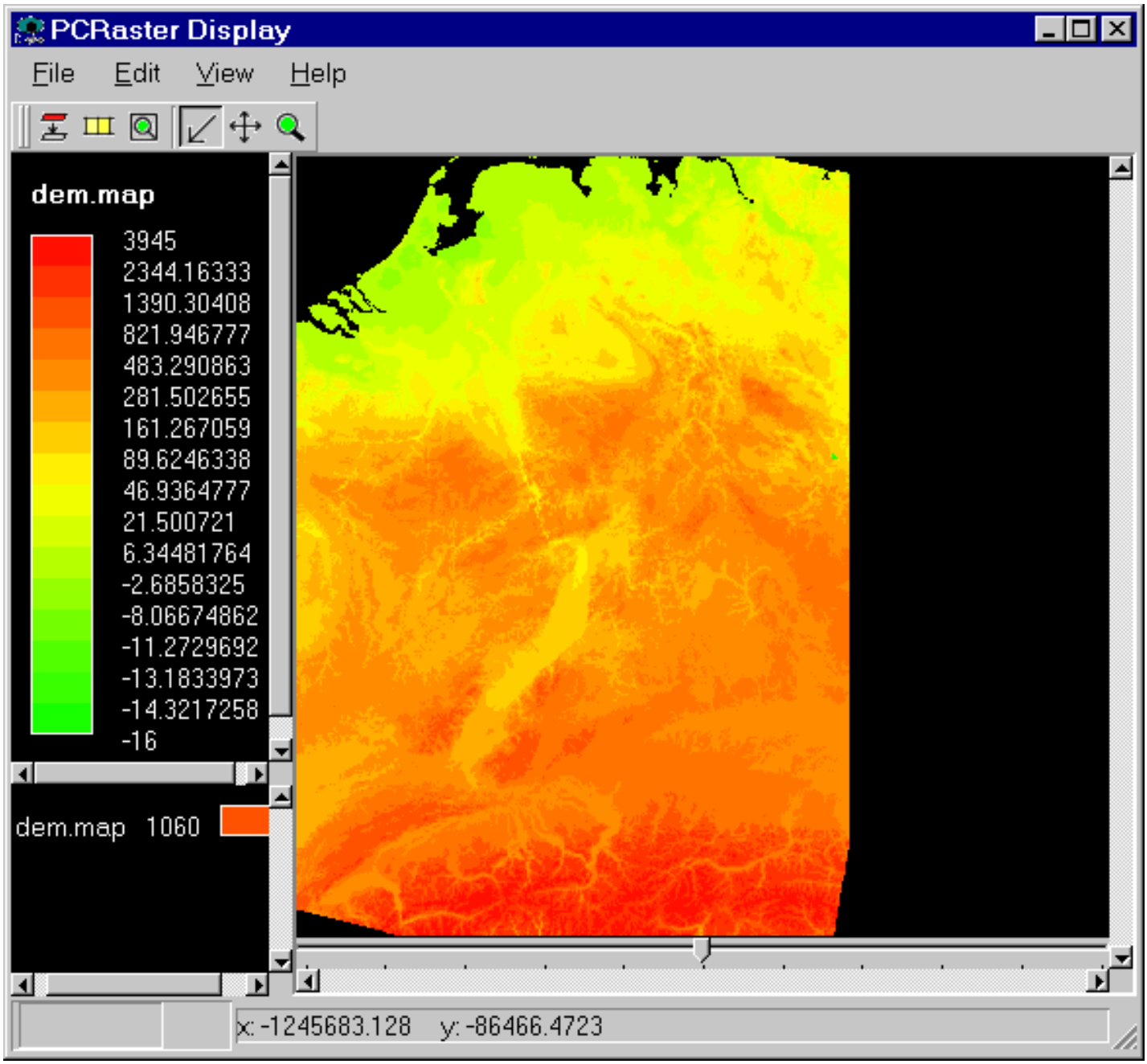
Screenshots of Display



1: Here a local drainage direction network is draped on top of an elevation model. The cursor view shows the flow direction (north) and height (90.15) at the location of the mouse.



2a: This is an elevation model of the rhine catchment. This example shows the default way Display visualizes a scalar map layer. The total range in data values is devided in 16 equaly sized classes. In the above case, all but the lowest classes are situated in the Alps. Compare this with figure 2b.



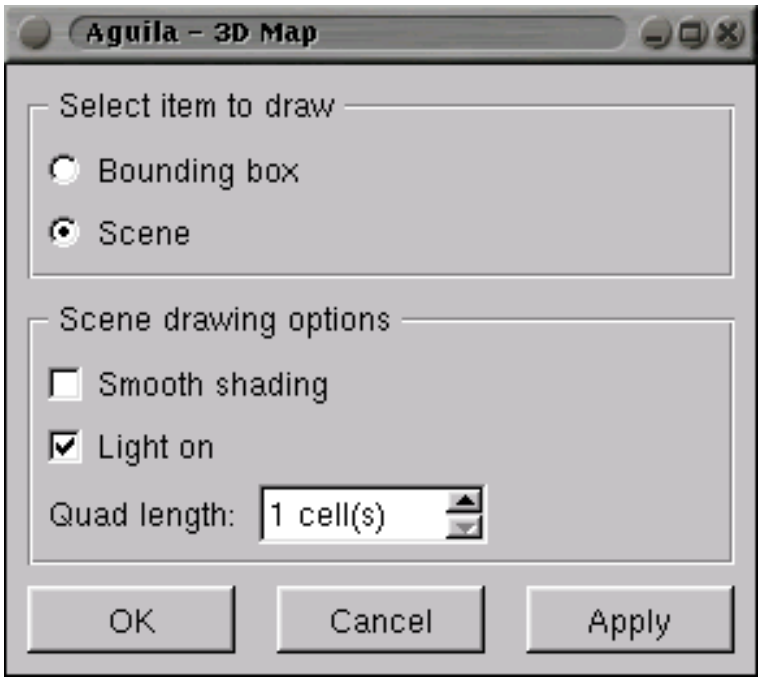
2b: To get a beter sense of the variability in the data values we need a way to assign more colours to lower regions. This is a situation where you can use the logarithmic or shifted logarithmic classification method. Since we have values below and equal to 0 we can't use the pure logarithmic classification. You can find these options in the properties dialog of the drawer.

Clickpath: Screenshots Versions: page: 20 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

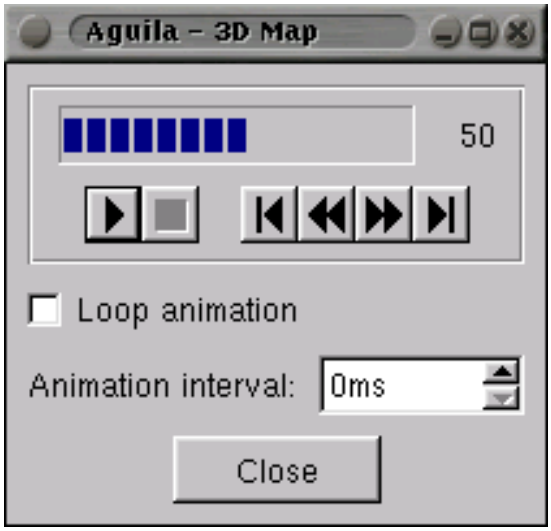
Department of Physical Geography Faculty of Geosciences Utrecht University The Netherlands

Universiteit Utrecht

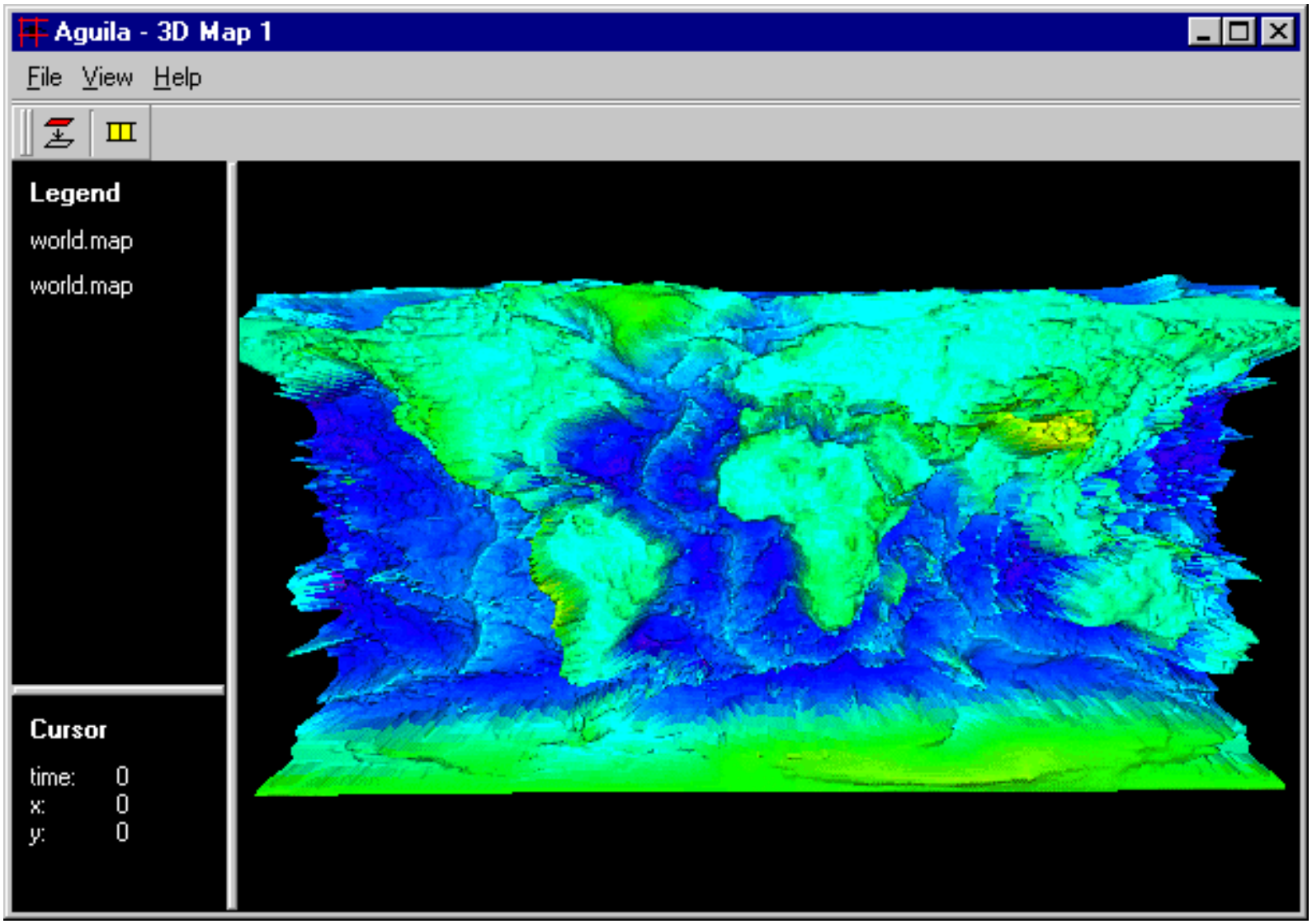
Screenshots



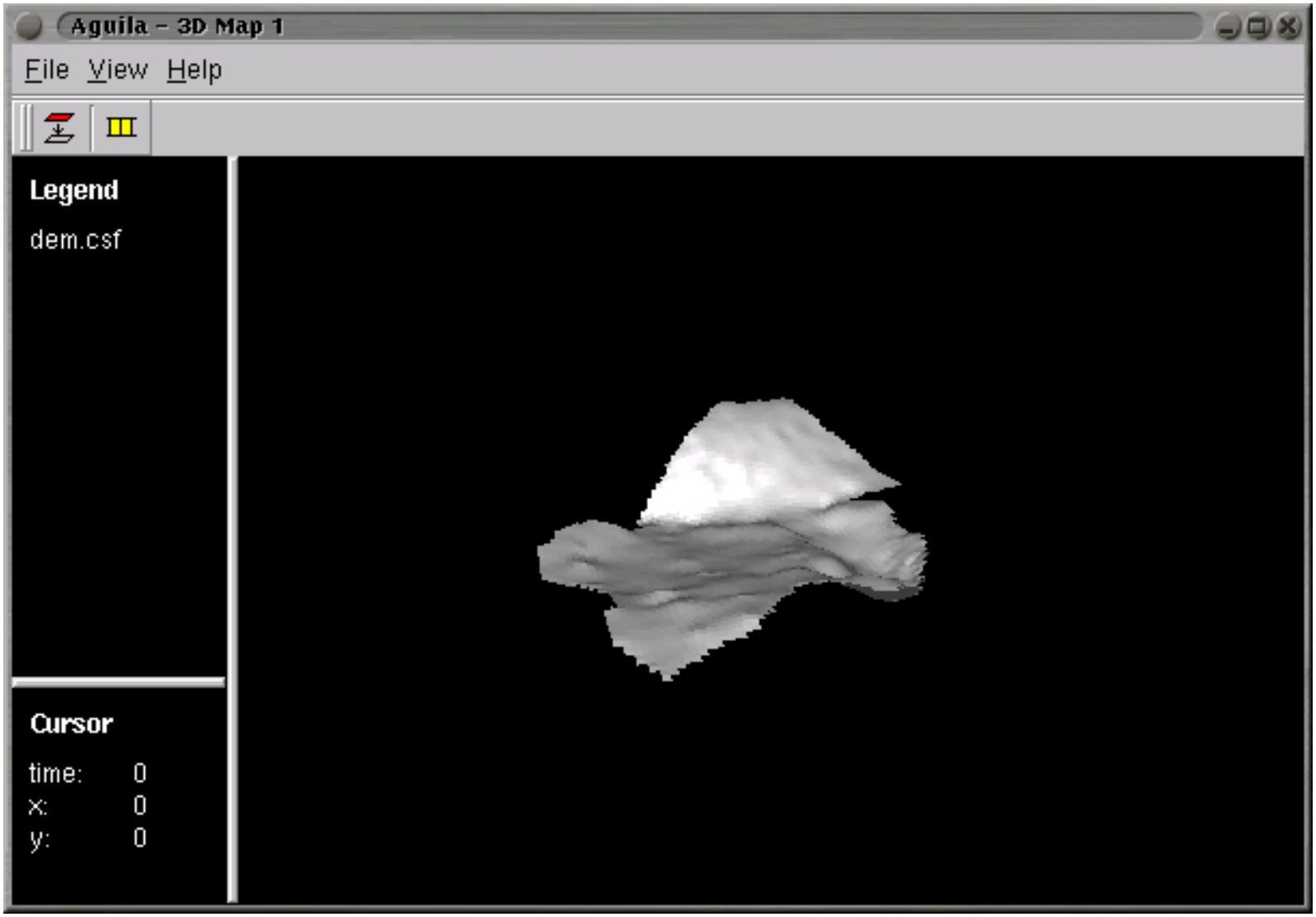
Properties dialog of the 3D map view.



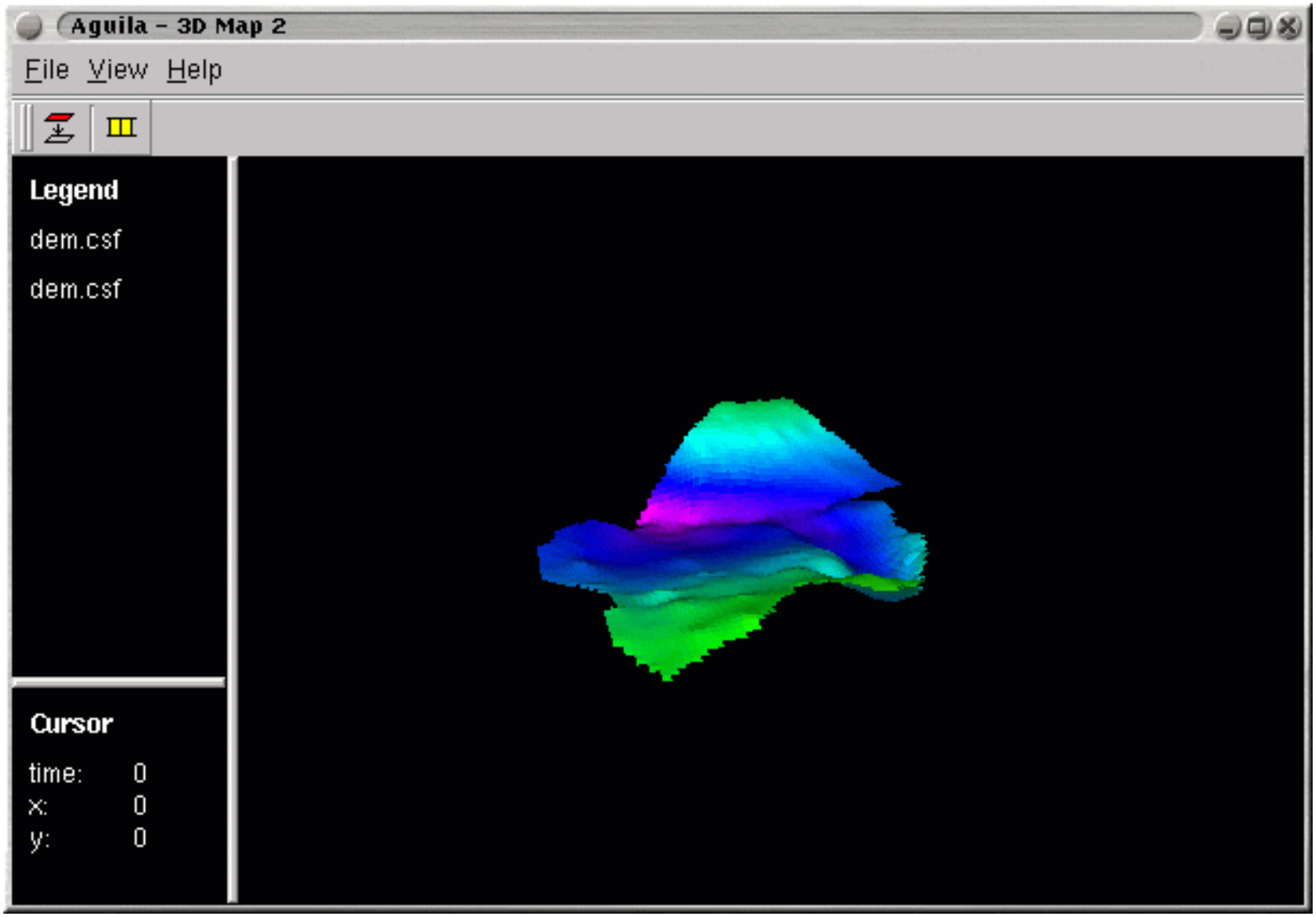
Animation control dialog.



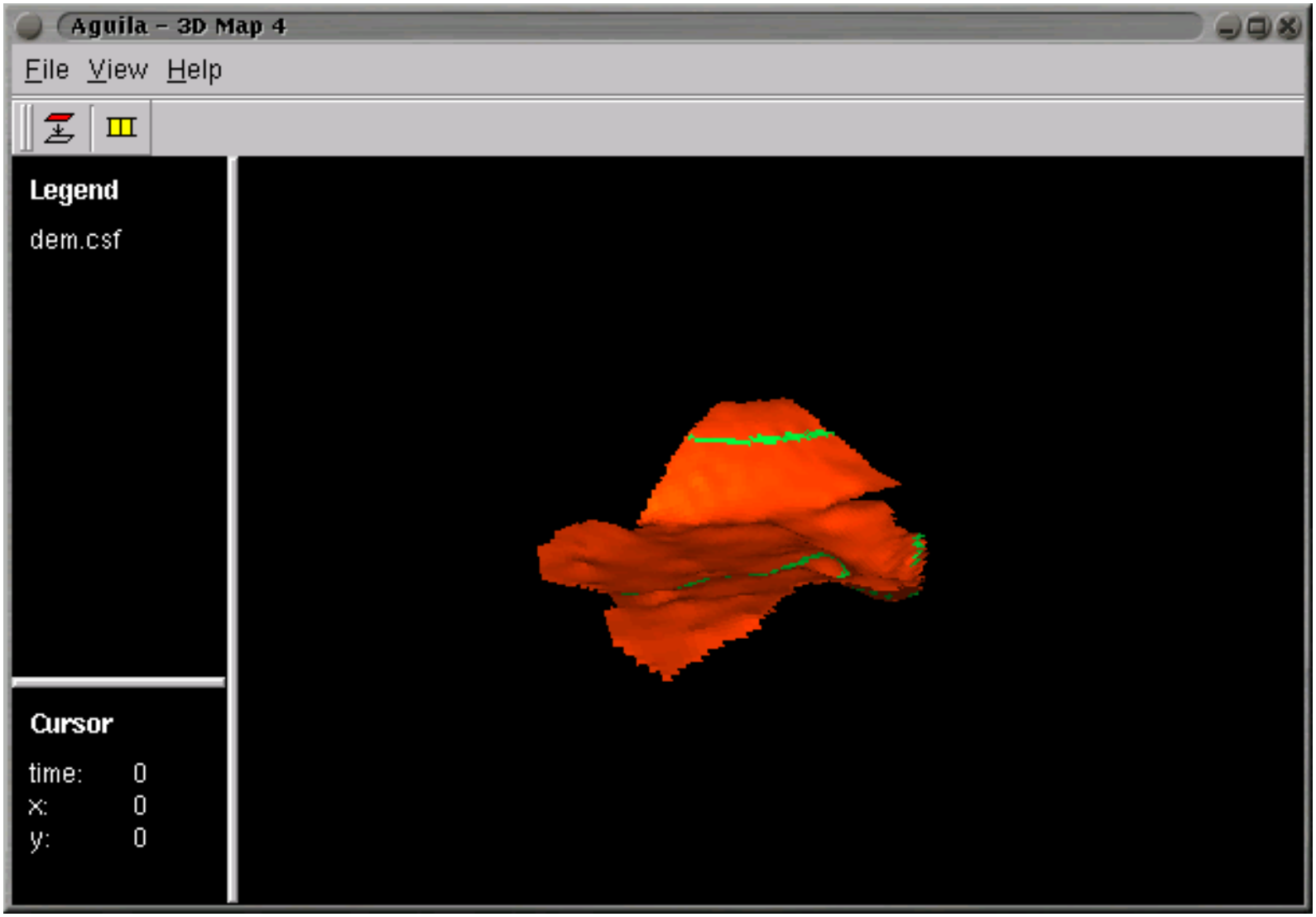
3D map of the world.



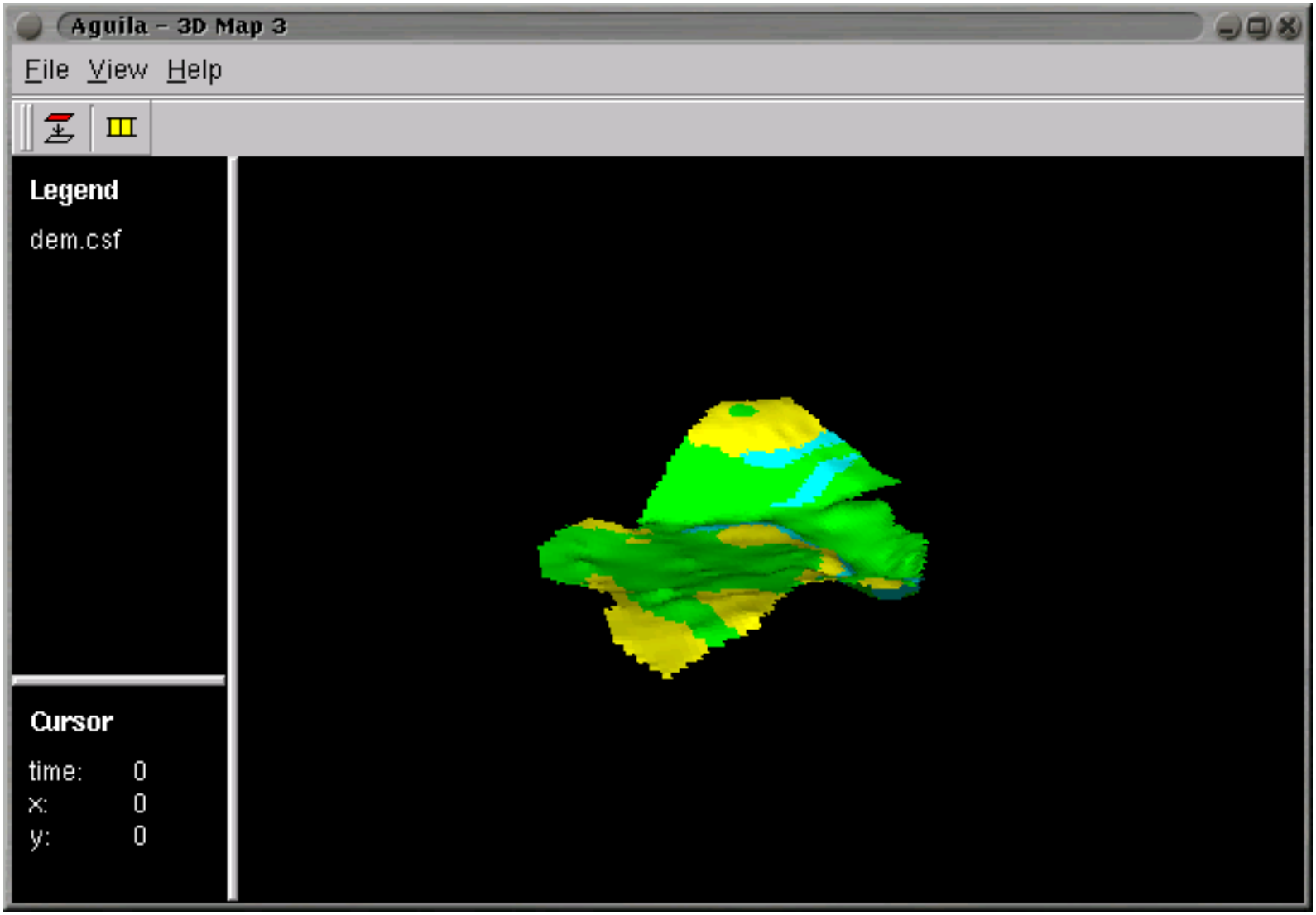
aguila dem.csf



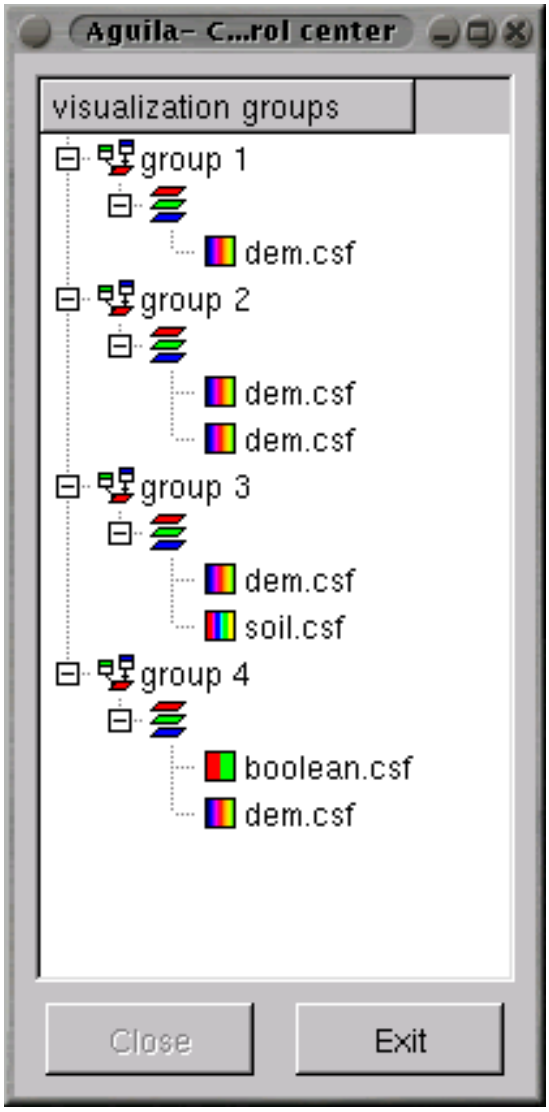
aguila dem.csf + dem.csf



aguila dem.csf + road.csf



aguila dem.csf + soil.csf



Control center.

Clickpath: Screenshots Versions: page: 23 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

FAQ

1. What's the easiest way to compare two different layers (timesteps) from the same dynamic stack? answer
2. How can I improve the performance of the camera and scene controls? answer

1. **What's the easiest way to compare two different layers (timesteps) from the same dynamic stack?**

   Since a visualization group only has one cursor position you can't show a layer from timestep t1 in a visualization and a layer from timestep t2 in another visualization if both visualizations are members of the same group. The easiest way to compare two different layers from the same dynamic stack is to copy the visualization. A new visualization group will be created, with the same data and the copied visualization. Now you have two groups with the same data. You can use the animation dialogs of both visualizations to select the layers from the stack you want to compare.

   Alternatively you can use the control center to 'manually' create a new visualization group, add data to it and add a visualization to it.

   The idea is that if you want to compare data on different cursor positions, you need different visualization groups.

2. **lHow can I improve the performance of the camera and scene controls?**

   Enlarge the quad length (press Shift-Q or use the properties dialog) before rotating or moving the camera or scene. Once you're in a good viewing position you can set the quad length back to 1 for a high resolution map.

Clickpath: FAQ Versions: page: 23 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

Common problems (and their solutions)

Colour palettes

Colour palettes

Problem:
I've selected some nice colours for a certain map layer. How can I override the default colour palette used by Display?

Solution:
While reading a map layer, Display checks if it has an in-file colour palette. If so, it reads and uses that palette. If not, it uses the internal, default colour palette. This procedure can be overriden by the -p commando line option of Display, e.g.:

```
display -p nicecolours.pal soil.csf
```

In the above case the nicecolours.pal palette will be used even if soil.csf has an in-file colour palette. If you want a certain colour palette to be permanently coupled to a certain map layer you can save the layer and the palette by selecting the 'Save' or 'Save as...' menu items from the file menu. The next time you want to view the layer you don't have to use the -p option anymore.

Clickpath: Known problems Versions: page: 20 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

Known problems

This section contains a list of known problems in the software. Some have workarounds which are described too.

Control center

Problem: You can open new visualization groups, visualizations and data sets, but the control center doesn't enable you to add a specific data set to a visualization.

Workaround: Use the File|Open menu option of the visualization's interface to add a new data set to a visualization.

3DMap

Problem: The arrow keys are supposed to rotate the scene relative to the current camera position. Unfortunately this doesn't work yet. Currently the arrow keys rotate the scene relative to the original camera position (which is fine if you don't move the camera).

Problem: The map saved in the png format is sometimes slightly different than the map in the 3DMap window.

Workaround: If this happens, maximize the window and resize it back. If you save the map again, it will contain an exact copy of the map in the window.

Clickpath: Known problems Versions: page: 23 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

Bugs

- A legend can be too large for Display. If a map layer has many classes the legend becomes very long. This can result in strange behaviour of the legend.
- If an ordinal layer is shown and the user wants to select another palette, then Display shows the wrong palettes from which the user can choose.
- If you save a layer more than once (File|Save as), than the created file will be badly formatted and useless. If you use (File|Save as) once, than you will not notice this bug.

Clickpath: Bugs Versions: page: 23 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

PCRaster Version 2 Manual

Table of Contents

1. Introduction to the PCRaster Package: Concepts, Package Layout
    1.1. Introduction
    1.2. GIS and Cartographic Modelling
    1.3. Dynamic Modelling
    1.4. gstat module: Geostatistical Modelling
    1.5. ADAM module: error propagation
    1.6. Hardware and software requirements, installation
2. The Database
    2.1. Introduction
    2.2. Concepts, kinds of data used in the database
    2.3. PCRaster maps
        2.3.1. Introduction
        2.3.2. Location attributes, missing values
        2.3.3. Data types
        2.3.4. Legends
    2.4. Tables
        2.4.1. Introduction
        2.4.2. Format
    2.5. Time series
        2.5.1. Introduction
        2.5.2. Format
    2.6. Point data column files
        2.6.1. Introduction
        2.6.2. Format
3. How to Import or Export Data, Display Maps, Global Options
    3.1. Introduction
    3.2. PCRaster maps: database management
        3.2.1. Creation of a PCRaster map; data import
        3.2.2. Data export from a PCRaster map

3.2.3. Cutting or joining maps; changing geographical location attributes or data types

3.2.4. Attaching a legend to a PCRaster map

3.2.5. Screen display, hard copy output of PCRaster maps

3.3. Tables: database management

3.3.1. Introduction

3.3.2. Creating and editing tables

3.4. Time series: database management

3.4.1. Introduction

3.4.2. Creating and editing time series

3.5. Point data column files: database management

3.5.1. Introduction

3.5.2. Creating point data column files, conversion to/from PCRaster maps

3.6. Global options and local options

3.6.1. Introduction, setting global options

3.6.2. Overview of global options

4. Cartographic Modelling

4.1. Introduction

4.2. General approach to Cartographic Modelling

4.3. Point operations

4.3.1. Introduction to point operations

4.3.2. Operators for point operations

4.4. Neighbourhood operations

4.4.1. Introduction

4.4.2. Window operations

4.4.3. Local drain direction operations

4.4.4. Friction paths

4.4.5. Transport of material over a ldd

4.4.6. Neighbourhood operations: visibility analysis

4.5. Area operations

4.5.1. Introduction

4.5.2. Operations over areas

4.6. Map operations

4.7. Command syntax and script files for cartographic modelling

4.7.1. Introduction

4.7.2. Command syntax

4.7.3. Script files

5. Dynamic modelling

5.1. Introduction

5.1.1. Concepts

5.1.2. The script

5.1.3. Timeinput and report in the script, running a script

5.1.4. How to make a dynamic model

6. Functional list of PCRaster operators

6.1. Point operators

6.1.1. Boolean operators

6.1.2. Comparison operators

6.1.3. Conditional statements

6.1.4. Missing value creation, detection, alteration

6.1.5. Relations in tables

6.1.6. Order

6.1.7. Maximize, minimize

6.1.8. Arithmic operators, trigonometric, exponential, logarithmic functions

6.1.9. Rounding

6.1.10. Data types: Conversion and assignment

6.1.11. Random number generation - cells

6.1.12. Coordinates, unique ID's

6.2. Neighbourhood operators

6.2.1. Windows operations

6.2.2. Derivatives of elevation maps

6.2.3. Spread operations

6.2.4. Operations with local drain direction maps

6.2.5. Operations for visibility analysis

6.3. Area operations

6.3.1. Operations over areas

6.3.2. Random number generation - areas

6.4. Map operations

6.4.1. Operations over maps

6.4.2. Random number generation - map

6.5. Time operations

6.5.1. Time operations

6.6. Data management

6.6.1. Random number generation - areas

6.6.2. Conversion of data

6.6.3. Cutting and joining together PCRaster maps

6.6.4. Generation of legends

6.6.5. Screen output

6.6.6. Hard copy output

Glossary

Reference Pages

I. List of PCRaster Operators

+ -- Addition

- -- Subtraction

/ or div -- Division

* -- Multiplication

** -- *n*th power of a first expression, where *n* is the value of a second expression

abs -- Absolute value

accucapacityflux, accucapacitystate -- Transport of material downstream over a local drain direction network

accuflux -- Accumulated material flowing into downstream cell

accufractionflux, accufractionstate -- Fractional material transport downstream over local drain direction network

accuthresholdflux, accuthresholdstate -- Input of material downstream over a local drain direction network when transport threshold is exceeded

accutriggerflux, accutriggerstate -- Input of material downstream over a local drain direction network when transport trigger is exceeded

acos -- Inverse cosine

and -- Boolean-AND operation

areaarea -- The area of the area to which a cell belongs

areaaverage -- Average cell value of within an area

areadiversity -- Number of unique cell values within an area

areamajority -- Most often occurring cell value within an area

areamaximum -- Maximum cell value within an area

areaminimum -- Minimum cell value within an area

areanormal -- Value assigned to an area taken from a normal distribution

areatotal -- Sum of cell values within an area

areauniform -- Value assignseds to area taken from an uniform distribution

asin -- Inverse sine

aspect -- Aspects of a map using a digital elevation model

atan -- Inverse tangent

boolean -- Conversion data type to boolean data type

catchment -- Catchment(s) of one or more specified cells

catchmenttotal -- Total catchment for the entire upstream area

cellarea -- Area of one cell

celllength -- Horizontal and vertical length of a cell

clump -- Contiguous groups of cells with the same value ('clumps')

cos -- Cosine

cover -- Missing values substituted for values from one or more expression(s)

defined -- Boolean TRUE for non missing values and FALSE for missing values

directional -- Data conversion to the directional data type

downstream -- Cell gets value of the neighbouring downstream cell

downstreamdist -- Distance to the first cell downstream

eq or == -- Relational-equal-to operation on two expressions

exp -- $Base_e$ exponential

fac -- Faculty or factorial of a natural positive number

ge or >= -- Relational-greater-than-or-equal-to operation

gt or > -- Relational-greater-than operation

idiv -- Quotient of integer division of values on first expression by values on second expression

if then -- Boolean condition determining whether value of expression or missing value is assigned to result

if then else -- Boolean condition determining whether value of the first or second expression is assigned to result

kinematic -- Dynamic calculation of streamflow through a channel

ldd -- Data conversion from specific data types to local drain direction data type

lddcreate -- Local drain direction map with flow directions from each cell to its steepest

downslope neighbour

lddcreatedem -- Modified digital elevation model

ldddist -- Friction-distance from the cell under consideration to downstream nearest TRUE cell

lddmask -- Local drain direction map cut into a (smaller) sound local drain direction map

lddrepair -- Reparation of unsound local drain direction map

le or <= -- Relational-less-than-or-equal-to operation

ln -- Natural logarithm ($_e$)

log10 -- Log $_{10}$

lookup -- Compares cell value(s) of one or more expression(s) with the search key in a table

lt or < -- Relational-less-than operation

maparea -- Total map area

mapmaximum -- Maximum cell value

mapminimum -- Minimum cell value

mapnormal -- Cells get non spatial value taken from a normal distribution

maptotal -- Sum of all cell values

mapuniform -- Cells get non spatial value taken from an uniform distribution

max -- Maximum value of multiple expressions

min -- Minimum value of multiple expressions

mod -- Remainder of integer division of values on first expression by values on second expression

ne or != -- Relational-not-equal-to operation

nodirection -- Expression of directional data type

nominal -- Data conversion data type nominal data type

normal -- Boolean TRUE cell gets value taken from a normal distribution

not -- Boolean-NOT operation

or -- Boolean-OR operation

order -- Ordinal numbers to cells in ascending order

ordinal -- Data conversion to the ordinal data type

path -- Path over the local drain direction network downstream to its pit

pit -- Unique value for each pit cell

plancurv -- Planform curvature calculation using a DEM

pred -- Ordinal number of the next lower ordinal class

profcurv -- Profile curvature calculation using a DEM

rounddown -- Rounding down of cellvalues to whole numbers

roundoff -- Rounding off of cellvalues to whole numbers

roundup -- Rounding up of cellvalues to whole numbers

scalar -- Data conversion to the scalar data type

sin -- Sine

slope -- Slope of cells using a digital elevation model

slopelength -- Accumulative-friction-distance of the longest accumulative-friction-path upstream over the local drain direction network cells against waterbasin divides

spread -- Total friction of the shortest accumulated friction path over a map with friction values from source cell to cell under consideration

spreadldd -- Total friction of the shortest accumulated friction downstream path over map with friction values from an source cell to cell under consideration

spreadlddzone -- Shortest friction-distance path over map with friction from a source cell to cell under consideration, only paths in downstream direction from the source cell are considered

spreadmax -- Total friction of the shortest accumulated friction path over a map with friction values from a source cell to cell under consideration

spreadzone -- Shortest friction-distance path over a map with friction from an identified source cell or cells to the cell under consideration

sqr -- Square

sqrt -- Square root

streamorder -- Stream order index of all cells on a local drain direction network

subcatchment -- (Sub-)Catchment(s) (watershed, basin) of each one or more specified cells

succ -- Ordinal number of the next higher ordinal class

tan -- Tangent

time -- Timestep

timeinput... -- Cell values per timestep read from a time series that is linked to a map with unique identifiers

timeinput -- Set of output maps per timestep with an extension that refers to the time at the timestep

timeoutput -- Expression value of an uniquely identified cell or cells written to a time series per timestep

timeslice -- Timeslice

uniform -- Boolean TRUE cell gets value from an uniform distribution

uniqueid -- Unique whole value for each Boolean TRUE cell

upstream -- Sum of the cell values of its first upstream cell(s)

view -- TRUE or FALSE value for visibility from viewpoint(s) defined by a digital elevation model

windowaverage -- Average of cell values within a specified square neighbourhood

windowdiversity -- Number of unique values within a specified square neighbourhood

windowhighpass -- Increases spatial frequency within a specified square neighbourhood

windowmajority -- Most occurring cell value within a specified square neighbourhood

windowmaximum -- Maximum cell value within a specified square neighbourhood

windowminimum -- Minimum value within a specified square neighbourhood

windowtotal -- Sum of values within a specified square neighbourhood

xcoordinate -- X-coordinate of each Boolean TRUE cell

xor -- Boolean-XOR operation

ycoordinate -- Y-coordinate of each Boolean TRUE cell

II. PCRaster applications

asc2map -- Converts from ascii file format to PCRaster map format

col2map -- Converts from column file format to PCRaster map format

legend -- Attaches a legend to or changes the legend of one or more maps.

map2asc -- Converts from PCRaster maps format to ascii file format

map2col -- Converts from PCRaster map format to column file format

mapattr -- Create a new PCRaster map, change or display location attributes of existing PCRaster map

resample -- Cuts one map or joins together several maps by resampling to the cells of the result map.

table -- Creates on basis of one or more maps a table with a score for each key in the table.

timeplot -- Plots timeseries in a x,y lineplot on the computer screen.

Index

List of Tables

2-1. List of data types, domaines for default cell representation, without legends

2-2. Example of a column table. The first, second and third column give the values of expression1, expression2 and expression3 respectively; the fourth column contains the value fields.

2-3. Example of a matrix table. The fields in the first row contain values of expression1; the fields in the first column contain values of expression2. The field in the top left corner is a dummy field. The remaining fields are value fields.

1. Cross table of the AND operator.

1. Cross table of the OR operator.

1. Cross table of the XOR operator.

List of Figures

1-1. The cell: relations between attributes within the cell and relations in lateral directions with attributes stored in neighbouring cells.

2-1. A stack of PCRaster maps resulting in a 2.5 D representation of the landscape. One cell is shown; its property is defined by the attribute values stored in map layers Map1, Map2, Map3,...

2-2. A table defining relations between PCRaster map layers; using these conditions a NewMap is generated, on a cell by cell basis

2-3. Location attributes used to define the spatial characteristics of a PCRaster map. For explanation, see text.

2-4. Directions of ldd codes. A value 5 (not shown) defines a cell without local drain direction (a pit).

4-1. Generation of a NewMap as a result of maplayers Map1, Map2, Map3,..

4-2. Point operation. A new map is generated on a cell-by-cell basis. No lateral relations between cells are included.

4-3. Neighbourhood operations within a window.

4-4. Neighbourhood operations over a path. For each cell the NewMap value is calculated on the basis of Map1, Map2, Map3,... values on a path from a source cell.

4-5. Neighbourhood operations within the catchment of a cell. For each cell the NewMap value is calculated on the basis of Map1, Map2, Map3,... values in the catchment of the cell, defined by the local drain direction network.

4-6. Path from a source cell to a target cell, crossing cell A.

4-7. Material transport over a local drain direction network. Bottom: Local drain direction map defining pattern of transport through neighbouring cells. Top: system diagram of an open system representing one cell.

4-8. Area operations. The cell value is determined on basis of values of cells which are in the same area as the cell under consideration.

4-9. Map operations. A non spatial value is calculated on basis of an aggregate value of a map or maps.

5-1. Schematic representation of a dynamic model.

5-2. Change of Soil moisture content (mm) with time, timestep one day. Included process: evapotranspiration. MoistInit: moisture content at start of model run (mm); MoistCrit: critical moisture content (mm); *E*act: actual evapotranspiration (mm/day).

5-3. Change of Soil moisture content (mm) with time, timestep one day. Included process: evapotranspiration and rain. MoistInit: moisture content at start of model run (mm); MoistCrit: critical moisture content (mm); *E*act: actual evapotranspiration (mm/day). Rain falls at time = 6 and 7 days.

1. Direction of ldd values. A value 5 (not shown) defines a cell without a local drain direction (a pit)

2. A sprinkling of the few definitions used in theory of pit removing.

List of Examples

2-1. Example of a time series file with a header. It gives the temperature at three weather stations, meant

for input or the output of a model with starttime 1, endtime 8 and timeslice 1.

2-2. Example of a point data column file in simplified Geo-EAS format.

4-1. Example of a cartographic modellinge script file

5-1. Example of a dynamic model.

5-2. Dynamic modelling script for change in soil moisture content; included process evapotranspiration.

5-3. Dynamic modelling script. Included processes: evapotranspiration and infiltrating rain.

5-4. Time series Rain.tss: rain at the study site.

1. Example of a column table. The first, second and third column give the values of *expression1*, *expression2* and *expression3* respectively; the fourth column contains the value fields.

2. Example of a matrix table. The fields in the first row contain values of *expression1*; the fields in the first column contain values of *expression*. The field in the top left corner is a dummy field. The remaining fields are value fields.

1. Example of a time series file with header

1. Example of a column table generated with the **table** operation with two input maps and the setting -n 4. The first and second column give the values of `PCRmap1` (data type scalar) and `PCRmap2` (data type nominal) respectively; the third column contains the score fields.

2. Example of a matrix table generated with the **table** operation with the settings --matrixtable and -n 4. Same input maps were used as in the first table (shown above). The fields in the first row contain ranges of `PCRmap1`; the fields in the first column contain values of `PCRmap2`. The field in the top left corner is a dummy field. The remaining fields are score fields.

Next
Introduction to the PCRaster Package: Concepts, Package Layout

Chapter 1. Introduction to the PCRaster Package: Concepts, Package Layout

Table of Contents

1.1. Introduction

1.2. GIS and Cartographic Modelling

1.3. Dynamic Modelling

1.4. gstat module: Geostatistical Modelling

1.5. ADAM module: error propagation

1.6. Hardware and software requirements, installation

1.1. Introduction

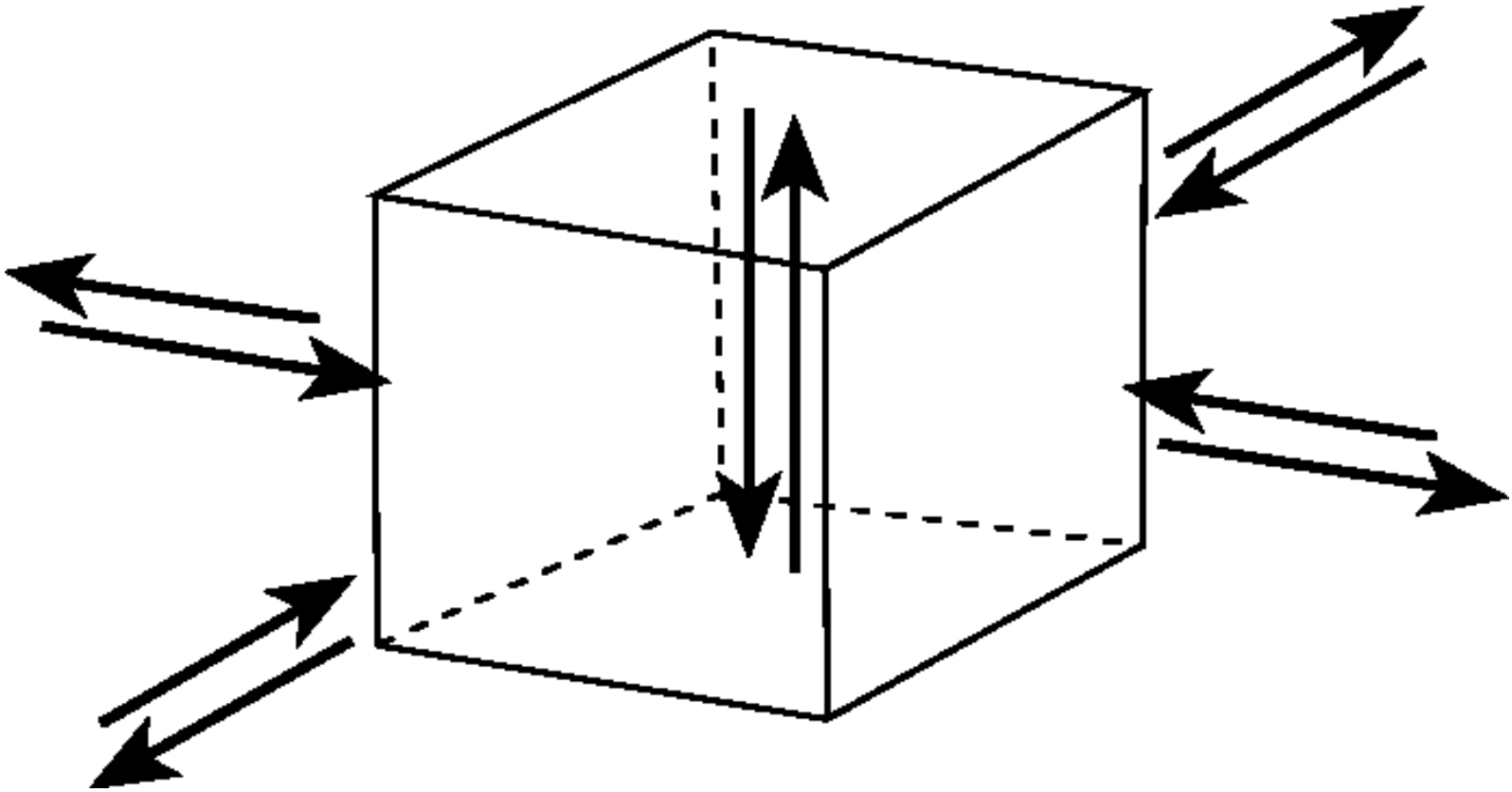
PCRaster is a Geographical Information System which consists of a set of computer tools for storing, manipulating, analyzing and retrieving geographic information. It is a raster-based system that uses a strict data type checking mechanism. This means that data type information is added to all spatial data, based upon the kind of attribute that the data represent. The use of data types controls the way the data are stored in the database and the possibilities for manipulation and analysis of the data. This strict data type checking mechanism has the advantage that it helps the user to organize the data and prevents the execution of operations that will make a nonsense result. The spatial data types implemented discriminate between various types of continuous fields and classified objects.

PCRaster has a relatively open database. The architecture of the system permits the integration of environmental modelling functions with classical GIS functions such as database maintenance, screen display and hard copy output. The modules for Cartographic and Dynamic Modelling are integrated with the GIS at a high level, which means that the GIS functions and modelling functions are incorporated in a single GIS and modelling language for performing both GIS and modelling operations.

The modules for geostatistical modelling (gstat module) and error propagation (ADAM module, not described in this manual) are integrated at a medium level with the GIS part of the package: both use a separate set of functions for manipulating the data; the map format of the central database is used and files can be automatically exchanged between the modules and the GIS part of the package. Exchange of ascii files with any other modelling or GIS package can be done using PCRaster conversion operations.

The central concept of PCRaster is a discretization of the landscape in space, resulting in cells of information. Each cell can be regarded as a set of attributes defining its properties, but one which can receive and transmit information to and from neighbouring cells. This representation of the landscape is often referred to as 2.5 D: the lateral directions in a landscape are represented by a set of neighbouring cells making up a map; relations in vertical directions, for instance between soil layers, are implemented using several attributes stored in each cell. GIS operations or operations used in modelling can be regarded as functions that induce a change in the properties of the cells on the basis of the relations within cells (between attributes on one cell location) or between cells, see Figure 1-1. In PCRaster, each functional module of the package represents a group of operations that change the properties of the cells in a specific way. A short review of the modules will be described (Section 1.2 to Section 1.5) and after that the hardware requirements and how to install PCRaster on a computer will be explained (Section 1.6)

Figure 1-1. The cell: relations between attributes within the cell and relations in lateral directions with attributes stored in neighbouring cells.



1.2. GIS and Cartographic Modelling

The central module of the PCRaster system is the group of PCRaster operations where the operations for Cartographic Modelling are integrated at a high level with the GIS functions of the package. The main GIS functions supported are user interfaces (i.a. screen display, hard copy output), conversion of data with other GIS packages and database management. No digitizing and scanning functionalities are implemented, but data transfer to and from other GIS packages that support these functionalities is simple. Spatial data are stored in the database as *PCRaster maps*, this is a binary format used for representation of raster maps in PCRaster.

The Cartographic Modelling part consist of operators for the static analysis of maps. This set of operators follows the concept of Map Algebra and Cartographic Modelling. There are several versions of Map Algebra, all with different names, but the concept used in PCRaster is strongly related to the concept of the MAP package designed by Tomlin ([tomlin80], [tomlin90]) and the algebra used by [berry87b]. The Cartographic Modelling part consists of a set of primitive operators that induce a change in the properties of the cells, where the change in properties is calculated on the basis of some kind of dependency within cells (point operations) or between cells (neighbourhood operations, area operations, map operations). An extensive set of operators is available in the PCRaster system: several point operators (analytical and arithmetical functions, Boolean operators, operators for relations, comparison, rounding, field generation etc.), neighbourhood operators for calculations in moving windows (highpass filtering, edge filtering, moving averages, etc.), area operators for calculations within specified areas (for instance soil groups), operators for the calculation of cost paths. In the PCRaster package a rich suite of geomorphological and hydrological functions is available that goes behind the range of operations generally considered as Map Algebra. These include functions for visibility analysis, catchment analysis and routing of transport (drainage) of material in a catchment using interactively generated local drain direction maps and transport (routing) operations.

This set of operators is a computer language designed especially for spatial and temporal analysis. It is an algebraic language, which means that the PCRaster operations can be applied and combined in the same way as algebraic calculations. In general an operation is done by typing:

pcrcalc *Result* = PCRasterOperator(*PCRasterExpression*)

where **pcrcalc** activates the PCRaster operation shell and PCRasterOperator is one of the PCRaster operations resulting in the *Result*. The *Result* may be a map or a non spatial value. The operation is done on the map *PCRasterExpression*. This map is called an *Expression* because it may be a map, but it may also be a PCRaster operation or a set of operations that result in a map or a non spatial value. This means that several PCRaster operations may be nested in one command. For instance the maximum slope on a slope map generated on basis of the elevation map Elevation can be calculated in one command line using the **mapmaximum** and **slope** operators by typing:

```
pcrcalc Result = mapmaximum(slope(Elevation))
```

As explained above, the separate commands are applied from the command line. Additionally, by linking these commands into PCRaster scripts or programs, it is possible to perform a theoretically unlimited number of commands consecutively. This is usually referred to as (static) Cartographic Modelling. Cartographic Modelling does not have a concept of time: several operations are performed consecutively, but they do not necessarily represent a process over time: the operations performed represent one, static change in the property of cells.

Elements of GIS and Catrographic modelling, like the database of the PCRaster package (Chapter 2), GIS functions (Chapter 3) and Cartographic Modelling (Chapter 4) are described in the next chapters.

Prev
Introduction to the PCRaster Package: Concepts, Package Layout

Home
Up

Next
Dynamic Modelling

1.3. Dynamic Modelling

The central idea of Cartographic Modelling is the derivation of new cell attributes from these attributes already present, or from attributes of neighbouring cells. In Dynamic Modelling the principle of spatial modelling is elaborated further by adding the concept of time: new attributes are computed as a function of attribute changes over time.

The Dynamic Modelling module is integrated at a high level with the part of the package for GIS functions and Cartographic Modelling. It provides a meta-language within which the user can build a dynamic model with the operators that are also used for Cartographic Modelling. Extra operators are added for creation of iterations through time and the reading of time series. The dynamic modelling language can be used for building a wide range of models, from very simple (point) models up to conceptually complicated or physically based models for environmental modelling (for instance erosion models). A dynamic model developed in the PCRaster Dynamic Modelling module can carry out all steps performed in modelling with ordinary low level GIS integrated models such as MODFLOW, MICROFEM (i.e. validating and calibrating). It has the advantage that the model is integrated at a high level with the GIS: data exchange problems do not exist, because the database of the GIS *is* the database of the model and vice versa. So it is very easy to run models using distributed data sets imported, analyzed and manipulated with the GIS and Cartographic Modelling part of the PCRaster package or created with the other modules of the package. Additionally the results of model runs can be visualised and analyzed without further data exchange. Chapter 5 covers the Dynamic Modelling module.

1.4. gstat module: Geostatistical Modelling

Gstat is the module of PCRaster for Geostatistical Modelling. Gstat is a seperate application, for details look at www.gstat.org. Most Likely your PCRaster installation came with a working version of gstat and seperate manual. Here we give a short description of geostatistical modelling and the functionality of gstat.

In geostatistical modelling it is assumed that the property of the cells is a realization of a spatial random function. It includes modelling the spatial dependence on basis of the known cell values and spatial prediction where values are predicted at cells with an unknown value using cells with a known value. In gstat, modelling the spatial dependence is done by estimating the variogram, the (pseudo) cross variogram, covariogram or cross covariogram and fitting (nested) variogram models (with interactive graphical display). Tools for spatial prediction are simple, ordinary or universal, univariable or multivariable, point or block kriging or conditional simulation. Simple inverse distance functions are also available.

The gstat module is integrated at a medium level with the GIS part of the package. It is a separate module, but conversion of data with the central database is simple. For interpolating point data, it uses a point data column format also used in Geo-EAS (Section 2.6). This point data format can easily be converted to PCRraster map format. The output from gstat of spatial data is in PCRaster map format: when performing interpolations in gstat PCRaster formatted maps can be used as a mask to specify the area over which interpolations are done and the resolution of the interpolated map. The resulting interpolated maps are in PCRaster map format and can be visualized and analyzed using the PCRaster operators.

1.5. ADAM module: error propagation

The ADAM module for error propagation is described in a separate manual, see [wesseling93]. Here we give a short description.

The ADAM module calculates the propagation of errors in a Cartographic Model. On the basis of the errors in the (spatial) input parameters of the Cartographic Model and non spatial or spatial correlation between these input parameters the ADAM module calculates the error in the output of the Cartographic Model (i.e. variance, skewness and quantiles of the output distribution). Error propagation techniques implemented are Monte Carlo method, first and second order Taylor method and Rosenblueth's method.

The error in the input parameters and the correlation between the input parameters is specified in a ADAM script file with the variogram models which are also used in the gstat module of PCRaster. ADAM uses the PCRaster map format for the input and output of the error propagation models.

2.6. Point data column files

2.6.1. Introduction

The concept of point data was discussied in a previous section of this chapter (Section 2.2). The next section (Section 2.6.2) gives the format used for point data column files. The creation of a point data column file and the conversion between point data column files and PCRaster maps will be discusse in the next chapter (Section 3.5).

2.6.2. Format

Ascii formatted column files are used for representation of point data in PCRaster. A column file consists of two columns containing the x and y coordinates respectively and one or more columns containing data values. Two types of column files can be used in PCRaster: a *column file in simplified Geo-EAS format* or a *plain column file* . These have the following format: 1) *column file in simplified Geo-EAS format*: line 1: header, description line 2: header, number (*n*) of columns in the file line 3 up to and including line *n* + 2: header, the names of the *n* variables subsequent lines: data which are formatted in at least three columns containing the x coordinates, y coordinates and values, respectively. Each line contains a record. The separator between the columns may be one or more whitespace character(s) (spaces, tabs) or ascii character(s). 2) *plain column file*: This is a file formatted like the simplified Geo-EAS format, but without header lines. The column separator may be chosen by the user. Fields with the x coordinates, y coordinates and values in the columnfile may contain the characters: -eE.0123456789. Fields may not be empty, valid fields are for instance: 25.11, -3324.4E-12 (which represents -3324.4 x 10-12), .22 (which represents 0.22).

The Table below gives an example of a column file in simplified Geo-EAS format.

Example 2-2. Example of a point data column file in simplified Geo-EAS format.

```
pH data January 17
4
xcoor
ycoor
pHfield
pHlab
349.34  105.03  3.4  4.1
349.36  102.51  3.4  4.1
348.89  104.00  3.6  4.1
348.44  102.68  3.5  4.1
```

```
349.89  104.72  3.8  4.1
```

Prev Time series | Home Up | Next How to Import or Export Data, Display Maps, Global Options

1.6. Hardware and software requirements, installation

PCRaster runs under UNIX or MS-DOS 3.1 or higher. For the DOS version, a 386 machine or equivalent with coprocessor and 4 MB RAM is required. A VGA monitor is required for displaying, VGA colour is recommended.

For installation see the PCRaster professional edition installation sheet appended to the software.

Chapter 2. The Database

Table of Contents

2.1. Introduction

2.2. Concepts, kinds of data used in the database

2.3. PCRaster maps

2.4. Tables

2.5. Time series

2.6. Point data column files

2.1. Introduction

This chapter describes the database of the PCRaster package. After a general introduction with the concepts behind the structure and the components of the database (Section 2.2), the components of the database wll be described in further detail, with emphasis on the practical on the practical aspects (formats, data types for instance. First the structure of PCRaster maps, like location attributes, missing values and data types will be described (Section 2.3) after which the different types of formats of tables, time series and point data column files will be explained (Section 2.4, Section 2.5 and Section 2.6). Of these sections, you will need the time series part (Section 2.5) only if you want to use the module for Dynamic Modelling (see Chapter 5). How to manage the database (data import, export, conversions etc.) and how to perform other GIS functions is described in Chapter 3.

Reference Pages

J.K. Berry, 1987, *Beyond Mapping: Concepts, Algorithms, and Issues in GIS* , World Books, GIS World Inc. Fort Collins, Colorada, USA , 246pp .

V.T. Chow, D.R. Maidment, and L.W. Mays, 1988, *Applied Hydrology*, McGraw-Hill, 0-07-100174-3.

B.K.P. Horn, 1981, "Hill shading and the reflectance map", *Proceedings of the I.E.E.E.*, 69, 14.

K.V. Mardia, 1972, *Statistics of Directional Data*, London: Academic Press.

A.K. Skidmore, 1989, " A Comparison of Techniques for Calculating Gradient and Aspect from a Gridded Digital Elevation Model.", *International Journal of Geographical Information Systems* , 3, 323-334.

A.N. Strahler, 1964, *Quantitative geomorphology of drainage basins and channel networks, section 4-II. In: Handbook of Applied Hydrology (V.T. Chow, et al. (1988))*. , McGraw-Hill, New York USA , 4-39 4-76.

C.D. Tomlin, 1980, *The Map Analysis Package (draft manual)* , Yale School of Forestry and Environmental Studies, New Haven, Connecticut .

C.D. Tomlin, 1990, *Geographic Information Systems and Cartographic Modelling* , Prentice-Hall, Englewood Cliffs, New Jersey , 0-13-350927-3.

C.G. Wesseling, 1993, *ADAM, an Error Propagation Tool for Geographical Information Systems*.

L.W. Zevenbergen and C.R. Thorne, 1981, " Quantitative analysis of land surface topography", *Earth Surface Processes and Landforms* , 12, 47-56.

Glossary

A

angle

Location attribute: the angle is the angle between the horizontal direction on the PCRaster map and the x axis of the real world coordinate system.

See Also: location attributes.

area

A fundamental spatial unit of a PCRaster map consisting of one cell or a set of cells.

areamap section

The section in a dynamic modelling script that defines the location attributes of the maps used in the model.

See Also: location attributes, dynamic modelling script.

attribute

Property of a geographic object or location.

B

boolean data type

data type for attributes that only may have a value TRUE (cellvalue 1) or FALSE (cell value 0).

See Also: data type.

C

Cartographic Model

A model that computes new attribute values from those attribute values already present. It represents one static change in the property of cells.

cartographic modelling script

A script of a Cartographic Model.

See Also: Cartographic Model, script.

cell

The basic spatial element of a PCRaster map.

cell length

Location attribute. The length of cells on a PCRaster map which is the same in horizontal and vertical direction. Measured in the distance unit of the real coordinate system.

cell property

The combined information at one cell location stored in one PCRaster map or several PCRaster maps.

cell representation

The method of storage used in the computer for cell values of PCRaster maps.

column

A column of cells in vertical direction on a PCRaster map.

See Also: number of columns.

D

database

A collection of interrelated information, usually stored on a harddisk. The PCRaster database includes data stored as binary PCRaster maps, and ascii formatted tables, time series and point data column files.

data type

Data description attached to a PCRaster map. It defines the scale and the domain of the data stored in the map and as result the behaviour with respect to operations performed upon the map.

See Also: boolean data type, nominal data type, ordinal data type, scalar data type, directional data type, ldd data type.

Dynamic Model

A model that computes new attribute values as a function of attribute changes over time. It represents a change in the property of cells over time.

directional data type

Data type for continuous data with a direction.

See Also: data type, scalar data type.

dubble real

Optional cell representation for scalar and directional data type, cell values stored in computer as REAL8.

dynamic modelling script

Script of a Dynamic Model.

See Also: Dynamic Model, script.

dynamic section

The iterative section in a dynamic modelling script which contains **calc** operations that are consecutively performed at each timestep.

See Also: iterative section.

E

expression

A PCRaster map or an operation resulting in a PCRaster map.

I

initial section

Cartographic Modelling script; the section containing the consecutively performed **calc** operations which describe the Cartographic Model. Dynamic Modelling script; the section which defines the map values of maps used in the dynamic section at the start of a model run.

See Also: dynamic section, section.

input

The set of one or more PCRaster maps, tables, time series or point data column files which are used in an operation to generate the result of the operation.

iterative section

iterative section: a section in a dynamic modelling script that describes the temporal change in map values.

See Also: dynamic modelling script, dynamic section.

K

keyword

A primitive word used in a script written in the PCRaster modelling language. It is a word which has a special meaning in the PCRaster modelling language. It is typed in lower case.

See Also: PCRaster modelling language, script.

L

larger integer

large integer: Optional cell representation for nominal and ordinal data type; cell values are stored in computer as INT4.

See Also: cell representation, small integer.

ldd data type

Data type for maps that represent a local drain direction network.

See Also: data type, local drain direction network.

local drain direction network

PCRaster map representation of flow paths in a landscape. Each cell contains a pointer towards its downstream neighbour or no pointer in case it is a pit.

See Also: outlet point, pit.

location attributes

The information entities attached to a PCRaster map which give together all spatial properties of the map.

M

map

The collection of digital information about a part of the earth's surface. The kind of maps used in PCRaster is the PCRaster map.

module

A piece of the PCRaster package with a distinct functionality.

N

nominal data type

Data type for classified data without order.

See Also: data type, ordinal data type.

number of columns

Location attribute. The number of columns in a PCRaster map.

See Also: column.

number of rows

Location attribute. The number of rows in a PCRaster map.

See Also: row.

O

operation

One static manipulation of one or more database components with one operator or several operators nested in one operation resulting in one or more database components (PCRaster map, table, time series, point data column file).

operator

A primitive PCRaster 'function' of Map Algebra, Cartographic Modelling, Dynamic Modelling and GIS. It calculates a result on basis of one or more inputs. Both the result and the inputs may be a PCRaster map, table, time series or point data column file.

ordinal data type

Data type for classified data with order.

See Also: data type, nominal data type.

outlet point

The pit cell at the end of a downstream path from a cell in a local drain direction network.

See Also: local drain direction network, pit.

P

PCRaster database

The database of PCRaster; see database.

PCRaster map

One of the kind of data in the PCRaster database. Contains spatial data of one attribute encoded in the form of a regular grid of cells covering an area. Binary format.

PCRaster modelling language

The computer language provided by PCRaster for building Cartographic or Dynamic Models using a script.

See Also: script, Cartographic Model, Dynamic Model.

pit

A cell in a local drain direction network that only has neighbours pointing towards it and no neighbours at lower or equal elevation that it can point to.

See Also: local drain direction network.

point data column file

One of the kind of data in the PCRaster database. Contains ascii formatted point data (x,y coordinates with attribute value(s)).

projection

Location attribute. The projection of the real world co- ordinate system assigned to the PCRaster map. It is an x,y field (also used in basic mathematics). The x coordinates increase from left to right. The y coordinates increase from top to bottom or from bottom to top.

See Also: location attributes.

R

result

The set of one or more PCRaster maps, tables, time series or point data column files that are generated by an operation.

row

PCRaster map; a series of cells in horizontal direction.

See Also: number of rows.

S

scalar data type

data type for continuous data that do not represent a direction.

See Also: data type, directional data type.

script

An ascii formatted computer programme of a Cartographic or Dynamic Model written in the PCRaster modelling language.

See Also: PCRaster modelling language.

section

A separate part of a script identified by a section keyword.

See Also: section keyword, script.

single real

Default cell representation for scalar and directional data type; cell values are stored as REAL4 in computer.

See Also: cell representation, script.

section keyword

A keyword that identifies the start of a section.

See Also: keyword, section.

small integer

Cell representation for boolean and ldd data type, default for nominal, ordinal and ldd data type; cell values are stored as UINT1 in the computer.

See Also: cell representation, larger integer.

statement

One line in a section of a script terminated with a semi colon (;).

See Also: section.

T

table

One of the kind of data in the PCRaster database. Contains relations between PCRaster maps. Ascii formatted.

timeseries

One of the kind of data used in the PCRaster database. Contains a time series of (aggregated) cell values. Ascii formatted.

V

variable

A PCRaster map, table, time series or point data column file in a Cartographic Model or Dynamic Model. Unlike a keyword its name is chosen by the model builder and starts with an upper case.

I. List of PCRaster Operators

Table of Contents

+ -- Addition

- -- Subtraction

/ or div -- Division

* -- Multiplication

** -- *n*th power of a first expression, where *n* is the value of a second expression

abs -- Absolute value

accucapacityflux, accucapacitystate -- Transport of material downstream over a local drain direction network

accuflux -- Accumulated material flowing into downstream cell

accufractionflux, accufractionstate -- Fractional material transport downstream over local drain direction network

accuthresholdflux, accuthresholdstate -- Input of material downstream over a local drain direction network when transport threshold is exceeded

accutriggerflux, accutriggerstate -- Input of material downstream over a local drain direction network when transport trigger is exceeded

acos -- Inverse cosine

and -- Boolean-AND operation

areaarea -- The area of the area to which a cell belongs

areaaverage -- Average cell value of within an area

areadiversity -- Number of unique cell values within an area

areamajority -- Most often occurring cell value within an area

areamaximum -- Maximum cell value within an area

areaminimum -- Minimum cell value within an area

areanormal -- Value assigned to an area taken from a normal distribution

areatotal -- Sum of cell values within an area

areauniform -- Value assignseds to area taken from an uniform distribution

asin -- Inverse sine

aspect -- Aspects of a map using a digital elevation model

atan -- Inverse tangent

boolean -- Conversion data type to boolean data type

catchment -- Catchment(s) of one or more specified cells

catchmenttotal -- Total catchment for the entire upstream area

cellarea -- Area of one cell

celllength -- Horizontal and vertical length of a cell

clump -- Contiguous groups of cells with the same value ('clumps')

cos -- Cosine

cover -- Missing values substituted for values from one or more expression(s)

defined -- Boolean TRUE for non missing values and FALSE for missing values

directional -- Data conversion to the directional data type

downstream -- Cell gets value of the neighbouring downstream cell

downstreamdist -- Distance to the first cell downstream

eq or == -- Relational-equal-to operation on two expressions

exp -- $Base_e$ exponential

fac -- Faculty or factorial of a natural positive number

ge or >= -- Relational-greater-than-or-equal-to operation

gt or > -- Relational-greater-than operation

idiv -- Quotient of integer division of values on first expression by values on second expression

if then -- Boolean condition determining whether value of expression or missing value is assigned to result

if then else -- Boolean condition determining whether value of the first or second expression is assigned to result

kinematic -- Dynamic calculation of streamflow through a channel

ldd -- Data conversion from specific data types to local drain direction data type

lddcreate -- Local drain direction map with flow directions from each cell to its steepest downslope neighbour

lddcreatedem -- Modified digital elevation model

ldddist -- Friction-distance from the cell under consideration to downstream nearest TRUE cell

lddmask -- Local drain direction map cut into a (smaller) sound local drain direction map

lddrepair -- Reparation of unsound local drain direction map

le or <= -- Relational-less-than-or-equal-to operation

ln -- Natural logarithm ($_e$)

log10 -- Log $_{10}$

lookup -- Compares cell value(s) of one or more expression(s) with the search key in a table

lt or < -- Relational-less-than operation

maparea -- Total map area

mapmaximum -- Maximum cell value

mapminimum -- Minimum cell value

mapnormal -- Cells get non spatial value taken from a normal distribution

maptotal -- Sum of all cell values

mapuniform -- Cells get non spatial value taken from an uniform distribution

max -- Maximum value of multiple expressions

min -- Minimum value of multiple expressions

mod -- Remainder of integer division of values on first expression by values on second expression

ne or != -- Relational-not-equal-to operation

nodirection -- Expression of directional data type

nominal -- Data conversion data type nominal data type

normal -- Boolean TRUE cell gets value taken from a normal distribution

not -- Boolean-NOT operation

or -- Boolean-OR operation

order -- Ordinal numbers to cells in ascending order

ordinal -- Data conversion to the ordinal data type

path -- Path over the local drain direction network downstream to its pit

pit -- Unique value for each pit cell

plancurv -- Planform curvature calculation using a DEM

pred -- Ordinal number of the next lower ordinal class

profcurv -- Profile curvature calculation using a DEM

rounddown -- Rounding down of cellvalues to whole numbers

roundoff -- Rounding off of cellvalues to whole numbers

roundup -- Rounding up of cellvalues to whole numbers

scalar -- Data conversion to the scalar data type

sin -- Sine

slope -- Slope of cells using a digital elevation model

slopelength -- Accumulative-friction-distance of the longest accumulative-friction-path upstream over the local drain direction network cells against waterbasin divides

spread -- Total friction of the shortest accumulated friction path over a map with friction values from source cell to cell under consideration

spreadldd -- Total friction of the shortest accumulated friction downstream path over map with friction values from an source cell to cell under consideration

spreadlddzone -- Shortest friction-distance path over map with friction from a source cell to cell under consideration, only paths in downstream direction from the source cell are considered

spreadmax -- Total friction of the shortest accumulated friction path over a map with friction values from a source cell to cell under consideration

spreadzone -- Shortest friction-distance path over a map with friction from an identified source cell or cells to the cell under consideration

sqr -- Square

sqrt -- Square root

streamorder -- Stream order index of all cells on a local drain direction network

subcatchment -- (Sub-)Catchment(s) (watershed, basin) of each one or more specified cells

succ -- Ordinal number of the next higher ordinal class

tan -- Tangent

time -- Timestep

timeinput... -- Cell values per timestep read from a time series that is linked to a map with unique identifiers

timeinput -- Set of output maps per timestep with an extension that refers to the time at the timestep

timeoutput -- Expression value of an uniquely identified cell or cells written to a time series per timestep

timeslice -- Timeslice

uniform -- Boolean TRUE cell gets value from an uniform distribution

uniqueid -- Unique whole value for each Boolean TRUE cell

upstream -- Sum of the cell values of its first upstream cell(s)

view -- TRUE or FALSE value for visibility from viewpoint(s) defined by a digital elevation model

windowaverage -- Average of cell values within a specified square neighbourhood

windowdiversity -- Number of unique values within a specified square neighbourhood

windowhighpass -- Increases spatial frequency within a specified square neighbourhood

windowmajority -- Most occurring cell value within a specified square neighbourhood

windowmaximum -- Maximum cell value within a specified square neighbourhood

windowminimum -- Minimum value within a specified square neighbourhood

windowtotal -- Sum of values within a specified square neighbourhood

xcoordinate -- X-coordinate of each Boolean TRUE cell

xor -- Boolean-XOR operation

ycoordinate -- Y-coordinate of each Boolean TRUE cell

Prev Home Next

Reference Pages +

2.5. Time series

2.5.1. Introduction

The concept of time series was discussed earlier in this chapter (Section 2.2). The following section (Section 2.5.2) gives the format used for time series. Creating and editing a time series will be discussed later on (Section 3.4.2, as will the use of timeseries in dynamic modelling (Chapter 5).

2.5.2. Format

The contents and the format (number of rows) of a time series must match the dynamic model for which the time series is used, especially the time dimension of the model. For a description of the time dimension and the terms used, see Section 5.1.2.4. Two types of format for a time series are used: the *time series with a header* and a *plain time series* without header. Both are ascii formatted text. 1) *time series with a header* An example of this sort of time series is below. It has the following format:
line 1: header, description
line 2: header, number of columns in the file
line 3: header, time column description
line 4 up to and including line *n* + 3: header, the names of the *n* identifiers to which the second and following columns in the time series are linked.
subsequent lines: data formatted in rows and columns, where columns are separated by one or more spaces or tabs. Each row represents one timestep *i* at time t(*i*) in the model for which the time series is used or from which the time series is a report; the first row contains data for timestep *i* = 1, the second row for timestep *i* =2, etc. The first column contains the time t at the timesteps. At the first row which contains data for the first time step (*i* = 1) it is always the starttime t(1). For the following consecutive rows, the time in the first column increments each row with the timeslice dt of the model: in the *i*th row (*i*th timestep) the time is t(1) + (*i*-1) x dt. The remaining columns (column number 2 up to and including number *n*+1) contain values related to the *n* identifiers, where column number *i* is linked to the unique identifier value *i*-1. So, the second column contains values related to a unique identifier of 1, the third column contains values related to a unique identifier of 2 etc.

Example 2-1. Example of a time series file with a header. It gives the temperature at three weather stations, meant for input or the output of a model with starttime 1, endtime 8 and timeslice 1.

```
Temp., three stations
4
time
1
```

```
2
3
1  23.6  28  23.9
2  23.7  22  24.8
3  23.7  22  25.8
4  21.0  24  21.1
5  19.0  24  17.2
6  18.9  22  17.9
7  16.2  22  15.9
8  16.8  24  14.9
```

2) *plain time series* This is a file formatted like the time series file with header, but without the header lines.

Prev | Home | Next
Tables | Up | Point data column files

2.4. Tables

2.4.1. Introduction

The concept of tables specifying relations between PCRaster maps was discussed earlier in this chapter (Section 2.2). The following section (Section 2.4.2) describes the format used for tables. For creating and editing a table see Section 3.3.

Table 2-2. Example of a column table. The first, second and third column give the values of expression1, expression2 and expression3 respectively; the fourth column contains the value fields.

<2,>	3	<,12>	1
<,2]	3	<,12>	3
<2,>	14	<,12>	7
<,2]	14	<,12>	9
<2,>	14	8	4
<,2]	14	8	8

Table 2-3. Example of a matrix table. The fields in the first row contain values of expression1; the fields in the first column contain values of expression2. The field in the top left corner is a dummy field. The remaining fields are value fields.

-99	1	2	3	4
12	6.5	6.5	6	6
14	-4	-4	-4	-4
16	-13	-13	-12	-12

2.4.2. Format

Two formats for tables are used, a *column table* and a *matrix table*. By default, PCRaster uses column tables. If you want to specify relations between only two maps it is sometimes better to use matrices instead. This is done by setting the global option --matrixtable (for global options, see Section 3.6). The formats of tables are:

1) *column table* In the column table relations between the values of several maps *expression*1, *expession*2,...*expression*n are given, where an *expression* can be a PCRaster map or a computation with PCRaster operators resulting in a PCRaster map. For each combination of values of *expression*1,

*expression*2,...*expression*n a new value can be specified. Table Table 2-3 gives an example of a column table.

The column table is an ascii file that consists of a number of *n*+1 columns. The first *n* columns are key columns, where *n* is the number of maps. The key columns consist of key fields; each key field is one value or a range of values. The key fields in the first column are linked to cell values of *expression*1, the key fields in the second column to values on *expression*2, and so on, where the key fields in the *n*th column are linked to values on *expression*n. The last column (column number *n*+1) contains so-called value fields; these are new values that may be assigned to a new PCRaster map. Sometimes, if the **table** operator is used these will contain the number of cells (score) that match the key. Each row in the column table is called a tuple. Of course, it consists of *n* key fields and one value field.

The fields are separated by one or more spaces or tabs. The number of spaces or tabs does not matter. A value field is one single value. A key field is a single value, or a range of values, where a range of values is typed as: '[' or '<' symbol, minimum value, comma (, character), maximum value, ']' or '>' symbol. The minimum and maximum values are included in the range if square brackets ('[' and ']') are used, they are not included if point brackets ('<' or '>') are used. Omitting a value in the range definition means infinity. Ranges can be used for nominal, ordinal, scalar and directional data types. Values in keys are typed as an ordinary number (for instance 24.453) or by using *base* $_{10}$ exponentials (for instance 32.45e3 means 32450). Column tables may consist of as many tuples as needed. Remember that when linking maps with the operator **lookup** , for each cell the value field is assigned of the *first* tuple (from top to bottom) that matches the set of *expression*1, *expression*2,...*expression*n values of the cell.

2) *matrix table* A 2D matrix table contains the relations between two expressions *expression*1, *expression*2, where an *expression* is a PCRaster map or a computation with PCRaster operators resulting in a PCRaster map. Table Table 2-3 gives an example of a matrix table.

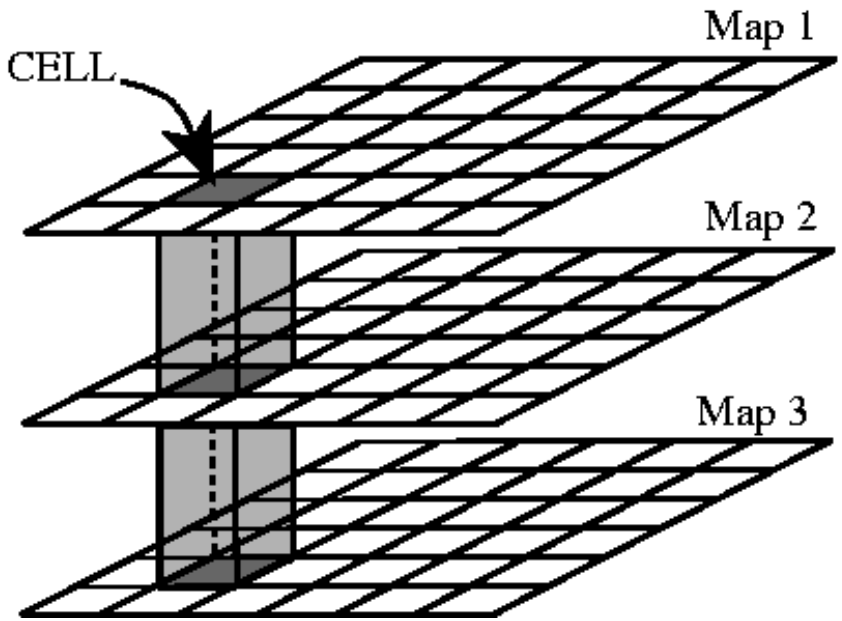
The matrix table is an ascii table with the following format. The first field in the top left corner of the matrix is not considered during PCRaster operations but is necessary to align the matrix; it is a dummy field and may have any value. The first row consists of this dummy field and the key fields which are linked to *expression*1. The first column consists of the dummy field and the key fields which are linked to *expression*2. The key fields may be one single value or a range of values, where a range is specified in the same way as it is done in a column table (see above). The remaining fields in the table are value fields and consist of the values which will be assigned to a new map. Or, if the **table** operator is used these will contain the number of cells (score) that match the key. In horizontal direction, fields must be separated by one or more spaces or tabs. All fields must be filled in.

2.2. Concepts, kinds of data used in the database

Four kinds of data are used in the PCRaster database. Data from 2D areas are represented by raster maps. These PCRaster maps have a special PCRaster format that enables simple and structured manipulation of spatial data in the package. It is the most important kind of data in the database: almost any PCRaster operation uses and/or generates a PCRaster map. For analysis of PCRaster maps with other software packages, conversion to ascii format is needed. The remaining three kinds of data (tables, time series and point data column files) represent relations between PCRaster maps, temporal data and data from points respectively. These kinds of data are in ascii format; as a result these can also be analysed with other software packages, without conversion.

In PCRaster a stack of *PCRaster map* layers represents the landscape, where each map layer represents one attribute, see Figure 2-1. The discretization of the spatial domain results in cells. At each cell location, the total information for that cell is represented by the values of the different layers at that cell. The representation described here is sometimes referred to as 2.5 D: the lateral directions are represented in real, while a certain kind of vertical dimension is implemented using several layers.

Figure 2-1. A stack of PCRaster maps resulting in a 2.5 D representation of the landscape. One cell is shown; its property is defined by the attribute values stored in map layers Map1, Map2, Map3,...



The spatial characteristics of a PCRaster map are defined by its geographical *location attributes* . These define the shape and the area covered by the map and the size of the cells.

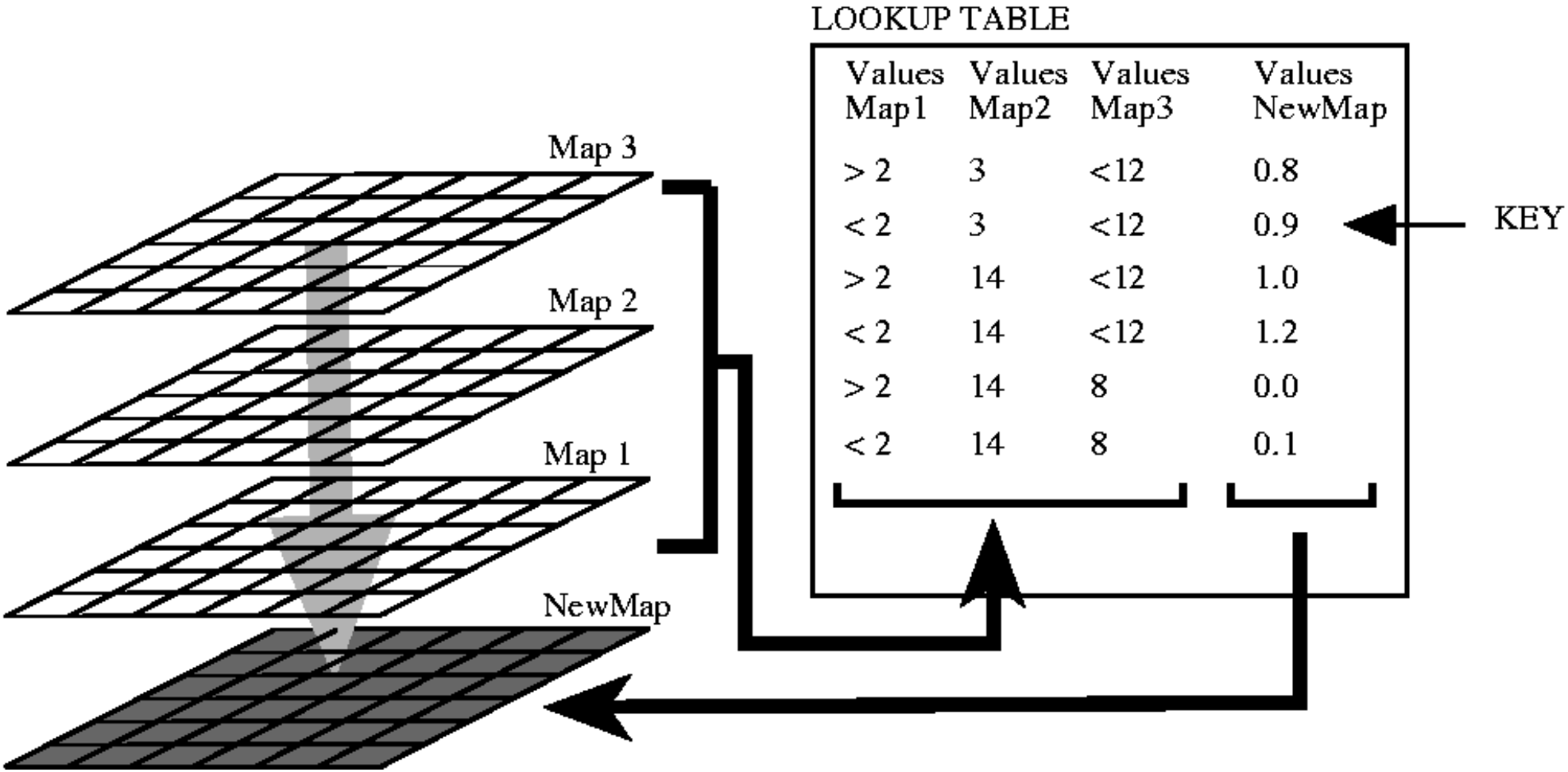
The kind of attribute represented by the layers controls the type of operations that can be done with the data stored in the layer. This knowledge is implemented in the PCRaster package by the idea of data types: each PCRaster map layer has a *data type* attached to it. Six data types are recognized. Data types for data in classes are the boolean, nominal and ordinal data types. The boolean data type is meant for data that may only have two values: TRUE or FALSE. Boolean logic can be applied to maps of this data type. The nominal data type represents data with an unlimited number of classes, for instance soil groups. The ordinal data type also represents data in classes; unlike the nominal data type it includes the concept of order between the classes, for instance classes that represent income groups. The scalar and directional data type

represent continuous data ; the scalar data type for data on a linear scale, for instance elevation, the directional data type for data on a circular scale, for instance aspect in the terrain. The ldd data type represents a map with a local drain direction network. For each cell, a local drain direction map contains a pointer to the neighbouring cell to which material (for instance water) will flow to. The direction of these pointers is represented by ldd codes.

Section 2.3 describes the format of maps, including the location attributes, data types and legends in detail.

Relations between PCRaster maps can be defined by *tables*, which is the second kind of data used in PCRaster, see Figure 2-2. In a table, map layers are combined by specifying keys. Each key gives a certain combination of cell values of the map layers 1,2,3,... A key may be for instance: the cell of map 1 must have a value 6, the cell of map 2 a value larger than 200 and the cell of map 3 must contain a negative value. Using the keys in a table a new map layer can be generated which contains information taken from several layers. For instance a soil map, vegetation map and a slope map can be combined using keys in a table containing the classes of these maps, generating a new map with landscape classes. Also a table can be used for determining the number of cells that match the conditions given in the keys. Section 2.4 describes the format of tables.

Figure 2-2. A table defining relations between PCRaster map layers; using these conditions a NewMap is generated, on a cell by cell basis



The third kind of data used in PCRaster is the *time series*. In Dynamic Modelling, time series are linked to a PCRaster map to control spatial data that vary over time and space: for each time step, a different spatial data set that represents a certain variable used in the model can be imported or stored. For instance when simulating

evapotranspiration of water in a catchment, for each time step the amount and the spatial distribution of rain water can be given in a time series; the amount of water that evaporates from a certain part of the map can be stored in a different time series. The time series is a table that crosses the unique identifier values on a PCRaster map with the numbers of the successive time steps used in the model. During a model run, it is read from top to bottom. If the time series is used for data input to the model, each unique identifier value on the PCRaster map is assigned the value linked to that unique identifier in the time series. This is done for each time step. If the time series is used to store data, for each time step the model results for certain areas specified on the PCRaster map can be assigned to the time series. Section 2.5 describes the format of time series.

In addition to spatial data in raster format, point data are used in the PCRaster package, stored as *point data column files*, the fourth sort of data used. Point data consist of a x,y coordinate and one or more attribute values. Quite often data will be available in this format, especially if they are gathered through field study. In the gstat module point data column files can be used for analysis of spatial structures with the variogram tools and for interpolation to a raster (in PCRaster map format) of estimated values using (block) kriging. Section 2.6 describes the format of point data column files.

Prev | Home | Next
The Database | Up | PCRaster maps

3.4. Time series: database management

3.4.1. Introduction

The format of time series has been described in an earlier chapter (Section 2.5). Underneath (Section 3.4.2) database management with time series is described.

3.4.2. Creating and editing time series

If you have time series data in a spread sheet program, database management program or a package for statistics you can create a PCRaster time series file by exporting your data as ascii formatted text. A PCRaster time series file must have the lay-out as described in Section 2.5. We advice to export the time series data in such a way that the resulting ascii formatted file will have a lay-out that looks like the lay-out used in PCRaster. Minor changes (for instance adding a header) can be made with a text editor.

Additionally, a timeseries file can be created by reporting a time series file to the database during a run of a dynamic model (see Section 5.1.3.3).

You can import a PCRaster time series file to an other software package by importing the time series as ascii text.

Chapter 5. Dynamic modelling

5.1. Introduction

Dynamic modelling is modelling of processes over time. In dynamic modelling new attributes are computed as a function of attribute changes over time. After the general concepts (Section 5.1.1) are explained, each part of the script will be described, from the structure in the seperate sections of the program (script) for a dynamic model (Section 5.1.2 to some special contents of a dynamic modelling script (timeinput, report) and the running of a script (Section 5.1.3). Finally, some examples of dynamic models are given (Section 5.1.4).

5.1.1. Concepts

5.1.1.1. Introduction

In a dynamic model, for each timestep a series of **pcrcalc** operations is consecutively performed using the resulting maps from the previous timestep and/or external data that define the value of an attribute for that timestep. This is done for all timesteps of a model run. Thus a dynamic model can be seen as a temporal sequence of static changes in the state of cells on map(s), each representing the change in the state of the modelled process over the timestep.

In the PCRaster database, the *time series* covers spatial data that vary over time. For each time step, it contains a set of map cell values that is input or output at that timestep. The format of time series will be described later on (Section 2.5). Other data used in Dynamic Modelling are the same as used for Cartographic Modelling: the *PCRaster map* and the **table**. In a dynamic model a PCRaster map is a variable that may have a different set of cell values for each time step: it is a map that changes in time. A table is used in dynamic modelling to define certain relations between maps. The table must be available in the PCRaster database before a model is run; a table cannot be changed or generated during a model run. The general concept and format of the time series, the PCRaster map en the table has been dicussed earlier (Chapter 2).

5.1.1.2. The sections of a script

Dynamic models are built with the language provided by PCRaster. Within this language the models can be programmed with the PCRaster operators of Cartographic Modelling. A *script*, which is a program written in the Dynamic Modelling language, consists of separate *sections*. Each section contains a certain functional part of the script. The division in sections is an essential concept of the Dynamic Modelling language. It tells the computer how to execute a program and it helps the user to structure the components of a model. The basic sections needed for building a model are the *binding* section, *areamap* section, *timer* section, *initial* section and the *dynamic* section. The Table below gives an example of a script for the simulation of the soil water balance. Here, you do not need to know the exact function of the model; it is only given as an idea of a script. The detailed description of the example in the Table will be discussed later on in this chapter (Section 5.1.4).

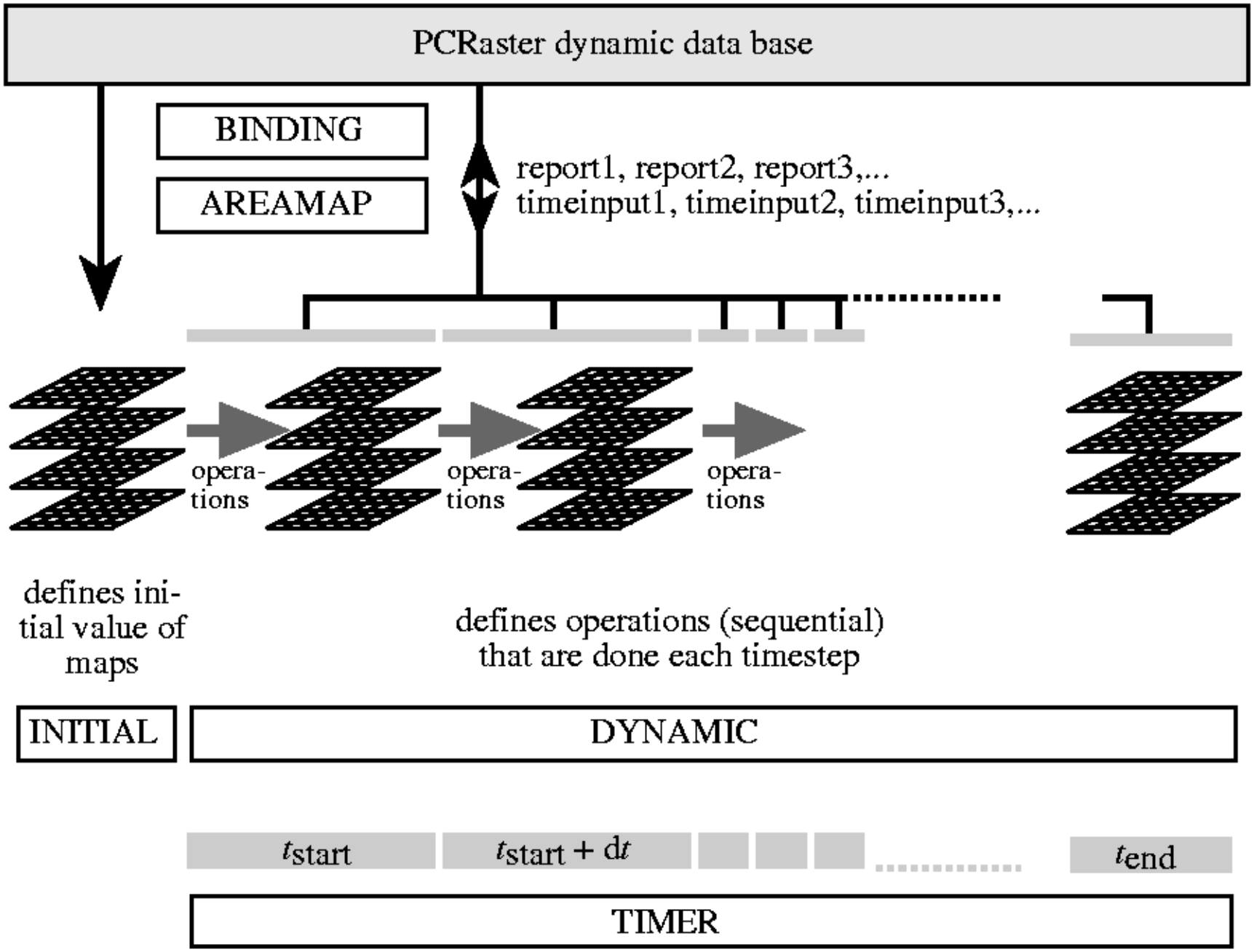
Example 5-1. Example of a dynamic model.

```
# model for calculation of reduction in soil moisture content
# incorporated processes: evapotranspiration and infiltration of rain
# timestep one day

binding
 MoistMeas = Moist952.map;
 MoistCrit = scalar(20);
 Eref = scalar(8);
 Kc = CrCoef95.map;
```

```
 TimeSeriesMax = Max8.tss;
 TimeSeriesMin = Min8.tss;
 RainTimeSeries = Rain.tss;

areamap
 Clone.map;

timer
 1 30 1;

initial
 report Eact = Kc * Eref;
 Moist = MoistMeas;

dynamic
 Rain = timeinputscalar(RainTimeSeries, 1);
 report Moist = Moist + Rain - Eact;
 MoistBelowCritial = Moist le MoistCrit;
 report TimeSeriesMax = mapmaximum(Moist);
 report TimeSeriesMin = mapminimum(Moist);
                              # reports to a time series
```

Figure 5-1 gives a conceptual picture of a Dynamic Model and its sections. The binding and areamap sections regulate the database management of the files used throughout the program. In principle, files mentioned in the Dynamic Modelling script are linked directly to those of the PCRaster database: the names used in the Dynamic Modelling script correspond with those of the database. The binding section defines a different names in the script: it binds the database file names to the names of a variable in the model.

Figure 5-1. Schematic representation of a dynamic model.



The areamap section defines the model area and modelling resolution. A clone map specifies the geographical location attributes of the maps used throughout the program. All maps used or generated by the model have the location attributes of the clone map in the areamap section.

The timer section controls the time dimension of the model. It specifies the duration of a model run by setting the time at the start (*t*(start)) and end (*t*(end)) of a model run. Additionally it specifies the time slice (d*t*) of the timestep. The number of timesteps of the model is the duration of the model run divided by the timeslice.

Spatial (maps) or non spatial attribute values at the start of the model run are given in the initial section. These values may be defined by one or more **pcrcalc** operations. The initial

section can be seen as a static Cartographic Modelling script which sets the initial attribute values used at the start of the dynamic section, for the iteration at the first timestep.

For each timestep *i*, the dynamic section defines the operations that result in the (map) values for timestep *i*. It is an iterative section that is repeated each timestep. The dynamic section consists of one or more **pcrcalc** operations that are performed sequentially each timestep. At the first timestep ($i = 1$), the dynamic section is run using the results from the initial section. The values that result from running the dynamic section at the first timestep are called the (map) values at timestep 1. The second timestep is run, starting with the results of timestep 1. These values are used for running the dynamic section at the third timestep ($i = 3$), etc. Thus, the operations performed during a timestep act upon the expressions that result from running the dynamic section at the previous timestep or upon an expression value that is the same for each timestep.

5.1.1.3. Timeinput: retrieving dynamic data from the database

For each timestep, timeinput assigns a new set of map values to an expression that is used in the dynamic section. Timeinput allows the import of specified data to the dynamic section at each timestep, irrespective of the results of the previous timestep: it defines an expression that is assigned a different set of cell values for each timestep. Each timestep, timeinput queries the database for a set of cell values especially meant for that timestep and assigns these to the expression.

Timeinput is done with a timeinput **pcrcalc** operation in the script. This is described in Section 5.1.3.2.

5.1.1.4. Reporting: storing dynamic data in the database

Reporting means storing model results in the PCRaster database. Both the results of an operation in the initial or in the dynamic section can be stored in the PCRaster database. If results of the dynamic section are reported, the results are stored in the database for each timestep. This can be done in two ways. First, the result of a **pcrcalc** operation can be stored in the database as a set of maps, where each map gives the values for a different timestep. Second, a time series can be used to report results: each timestep, the cell values of a given set of cells are written to a time series file.

Reporting of results is done with the **report** keyword. The use of this keyword in the script is given in Section 5.1.3.3.

5.1.2. The script

5.1.2.1. Introduction, layout of the script

This Section 5.1.2 describes the structure of a Dynamic Modelling script and its contents. The script is an ordinary ascii text file; an example was given in a Table in the beginning of this chapter. A script consists of separate sections, each with a defined function in the model. A dynamic model script contains the sections binding, areamap, timer, initial and dynamic; in this order. Each section starts with the *section keyword* of the section. The section keyword is followed by one or more *statements* that give the content of the section. Each statement is terminated by a semicolon (;) sign.

In principle, white space characters (spaces, empty lines) can be used anywhere in the script: all white space characters are ignored during execution of the script. For a structured script, we advise users to type the section keywords and the statements on separate lines. Remarks about the contents of the script are typed after a # character: everything typed on a line after this character is ignored and has no effect on the function of the model.

A statement in a section may contain *keywords*, names of variables, or numbers. Keywords are defined by the PCRaster Dynamic Modelling language and have a special meaning in the language. For instance the section keyword defines the start of a section in the script. Other keywords in a script are the PCRaster operators and for instance the **report** keyword. Keywords must always be typed in lowercase characters. Unlike a keyword, the name of a variable is chosen by the user. It may be the name of a PCRaster expression (a map or a non spatial value) or the name of a table or time series. To distinguish between keywords and names of variables, we strongly recommend that the name of a variable always begins with an uppercase character.

5.1.2.2. Binding section

In principle, if a variable is not mentioned in the binding section, the variable name in the script is directly linked to the corresponding file name in the PCRaster database: the file in the database that is used or generated during a model run has a corresponding name in the script and in the database. The binding section, identified by the section keyword **binding**, allows one to use a name for a variable in the script that is different from the file name of that variable in the database. This is because you probably may want to run a program a number of times, each time with a different set of data files and with a different set of resulting files. In most cases, these data files are used a large number of times throughout the program. Using

the binding, you need only fill in the names of the files you want to use as input names and output names for the model run in the binding section, without changing all the file names in the rest of the program. Both file names used as input files for the model and names that are stored in the database during a model run with the **report** keyword may be given in the binding section. If the PCRaster DOS version is used, an additional advantage of the binding section is that it allows one to get round the DOS rules for filenames (format 8.3: xxxxxxxx.xxx) in the script: the unpractical DOS names with a dot can be replaced by model names which may be as long as you want, with or without a dot.

In the binding section, the name of a file in the database is bound (linked) to its name in the model with the following statement:

NameInModel =
DatabaseFilename;

where *DatabaseFilename* is the file name under which the variable is available in the database or will be stored to the database and *NameInModel* is the name used for the variable throughout the script.

Alternatively, a constant value can be assigned to a variable with the statement. This applies only if ModelName is a PCRaster map which is input to the model:

NameInModel =
value;

where value is a number; *NameInModel* has this value throughout the program: its value cannot be changed in one of the other sections. It has no data type attached to it. Attaching a data type to the PCRaster map *NameInModel* with a constant value can be done using one of the data type assignment operators (**boolean**, **nominal**, **ordinal**, **scalar**, **directional**, **ldd**). An example which assigns a nominal data type to ClassMap is:

```
ClassMap = nominal(3);
```

The data type assignment operators are the only operations that can be given in the binding section, other operators cannot be used.

Not all variable names need to be mentioned in the binding; as above said, the filenames of the variables in the database can also be used directly in the script. If no variable names are bound at all, the section may be omitted from the model script.

5.1.2.3. Areamap section

The section keyword **areamap** defines the areamap section. It contains one statement: the name of a map which is used as clone map in the model, followed by a semi colon. This map may contain any kind of data; only its location attributes are of importance. All maps that are generated during a model run have the location attributes of the clone map. Also, all maps that are used as input to the model must have location attributes which correspond with the map in the areamap section.

5.1.2.4. Timer section

The timer section, identified by the section keyword **timer**, gives the time dimension of the model. It contains one statement, consisting of three values:

starttime endtime
timeslice;

The iterative part of the model is run between the starttime and the endtime. The timeslice defines the time between the consecutive timesteps.

This version of the Dynamic Modelling module of PCRaster imposes restrictions on the time dimension of a model. The starttime and timeslice cannot be chosen by the user and must always be 1. Only the endtime can be chosen, it must be a whole number larger than 1.

5.1.2.5. Initial section

The initial section, identified by the section keyword **initial**, is meant to prepare the set of input variables which are needed to run the dynamic section at timestep 1, the first time that the dynamic section is run.

The initial section can be compared with a Cartographic Modelling script. It contains a set of **pcrcalc** operations which are performed consecutively, from top to bottom in the section. Each line contains a **pcrcalc** operation and is concluded with a semi colon (;) sign. The resulting variables of the initial section are the initial values that are used as input variables for running the dynamic section at timestep 1.

It may be that initial variables (maps for instance) for running the dynamic section at timestep 1 are already present in the PCRaster database. These do not need to be generated or

changed in the initial section and can directly be used in the dynamic section, because they already have the correct value. On the other hand, *all* variables that are needed as input for running the dynamic section for the first time must either be defined in the initial section or must be already present in the database. This holds also for variables that have an initial value 0.

If the initial section is not needed to define the initial values of the variables it can be omitted from the script.

5.1.2.6. Dynamic section

The dynamic section, identified by the section keyword **dynamic**, contains **pcrcalc** operations that are performed at each timestep *i*. The operations are sequentially performed from top to bottom in the section. Each line gives a **pcrcalc** operation and is concluded with a semicolon (;) sign. At the start of running the dynamic at timestep *i*, variables have the value that results from running the dynamic at timestep *i* - 1. These values are used as input values for running the dynamic at timestep *i*. The values of the variables at the end of running the dynamic section at timestep *i* are the input values for running the dynamic section at timestep *i* + 1.

5.1.3. Timeinput and report in the script, running a script

5.1.3.1. Introduction

This Section 5.1.3 describes the contents of a script meant for timeinput of data (Section 5.1.3.2) and reporting results (Section 5.1.3.3). The last section (Section 5.1.3.4) describes how a script is run.

5.1.3.2. Operators for timeinput

Two PCRaster operators perform a timeinput operation: **timeinput** and **timeinput...** (e.g. **timeinputboolean**, **timeinputnominal**, etc.). These operations are used only in the dynamic section. Like the other PCRaster operators, the timeinput operators result in either a spatial or a non spatial expression. For each timestep, timeinput assigns specified cell values to the resulting expression, independent of the cell values at the previous timestep. The assignment of a different set of cell values to the expression for each timestep can be done in two ways:

The **timeinput** operator uses a set of maps in the database: the database must contain a PCRaster map with a file name extension referring to the timestep for which it is meant. The timeinput operator assigns to the expression in the dynamic section each timestep the map in the database with the extension referring to that timestep.

The **timeinput...** operators (**timeinputboolean**, **timeinputnominal**, **timeinputordinal**, **timeinputscalar**, **timeinputdirectional** and **timeinputldd**) use a time series. The time series is linked to a given expression with unique identifier cell values. For each timestep, the time series gives cell values for these unique identifiers. These cell values are assigned to the timeinput expression in the dynamic section on basis of the unique identifiers on the unique identifier expression.

For a detailed description of the timeinput operations, see **timeinput** and **timeinput...**. Section 2.5.2 gives the format of time series.

5.1.3.3. report keyword; and the timeoutput operator

The **report** keyword stores the result (which is always on the left hand side) of a **pcrcalc** operation to the database. Reporting is done by typing the keyword **report** before a **pcrcalc** operation:

report
VariableName = **pcrcalc**
operation;

for instance:

```
report NOStdDev = sqrt(NOVariance);
```

has the effect that the result (NOStdDev) of the **pcrcalc** operation is stored to the database. In a script, a VariableName cannot be used for report more than once.

The **report** keyword has no effect on the **pcrcalc** operation that is done, the only effect of **report** is that it stores the result of the operation in addition to computing it. If results of the iterative dynamic section are reported, the results are stored in the database for each timestep. The model results for each timestep can be reported in two different ways; the sort of report that is made depends on the sort of operation that is prefaced with the **report** keyword.

First, the results of an ordinary **pcrcalc** operation can be reported. If the result of the operation preceded with the **report** keyword is a map, each timestep the whole map is stored in the database with a filename that refers to the timestep under consideration. The file names of these maps in the database consist of two parts: the suffix and an extension. The suffix corresponds to the name of the variable that is prefaced by the **report** keyword. The suffix is followed by the time extension which corresponds with the time of the timestep at which the map is generated. The filename has a format that corresponds with the DOS rules for filenames: 8 characters, a dot and 3 characters. This is best illustrated by an example. Imagine a model with starttime *t*(1) = 1, endtime *t*(*i*= *i*end) = 1200 and timeslice d*t* = 1. In the dynamic section, the statement

```
report Snow = Snowfall * 2;
```

stores 1200 maps in the database with filenames Snow0000.001, Snow0000.002,...Snow0001.199, Snow0001.200. If this statement were put in the initial section only one map is stored in the database with filename Snow. Remember that it is not possible to report variable name Snow more than once: both in the initial and in the dynamic section.

Maps stored in the database with the **report** keyword have the filename format which is also needed for the **timeinput** operation.

If the result of the **pcrcalc** operation which is reported is a non spatial value it is automatically stored as a time series. At the end of the model run, the time series will contain the resulting non spatial value for each timestep.

The second way in which a report can be made is reporting results of a **pcrcalc** operation especially meant for reporting: the **timeoutput** operator. This operators always create a time series and must always be prefaced with the **report** keyword. They are only used in the iterative dynamic section. The **timeoutput** operator reports cell values of a specified cell or cells of an expression to a time series in the database. The result of one map operation resulting in a non spatial value is **always** written to a time series. It is meant to report statistics of a map for each timestep. Section 2.5.2 gives the format of time series.

5.1.3.4. Running a script

A Dynamic Modelling script is run in PCRaster (MS-DOS and UNIX) by typing:

pcrcalc -f
NameOfModel

where *NameOfModel* is the file name of the ascii formatted model script.

5.1.4. How to make a dynamic model

5.1.4.1. The soil moisture model with evapotranspiration

This following sections give some examples of a Dynamic Modelling script. We describe how the change in soil moisture content as a result of precipitation and evapotranspiration can be modelled using the PCRaster Dynamic Modelling language. In this section, we start with a script for a simplified model, without rain during a model run. The next section (Section 5.1.4.2) describes a somewhat more complicated model, incorporating rain modelled with a timeinput operation.

Evapotranspiration is the combined water loss to the atmosphere by evaporation from the soil and transpiration from plants. The actual evapotranspiration *E*act (mm/day) from a crop can be calculated with the relation ([chow88]):

*E*act =
*k*s x *k*cx
*E*ref
(1)

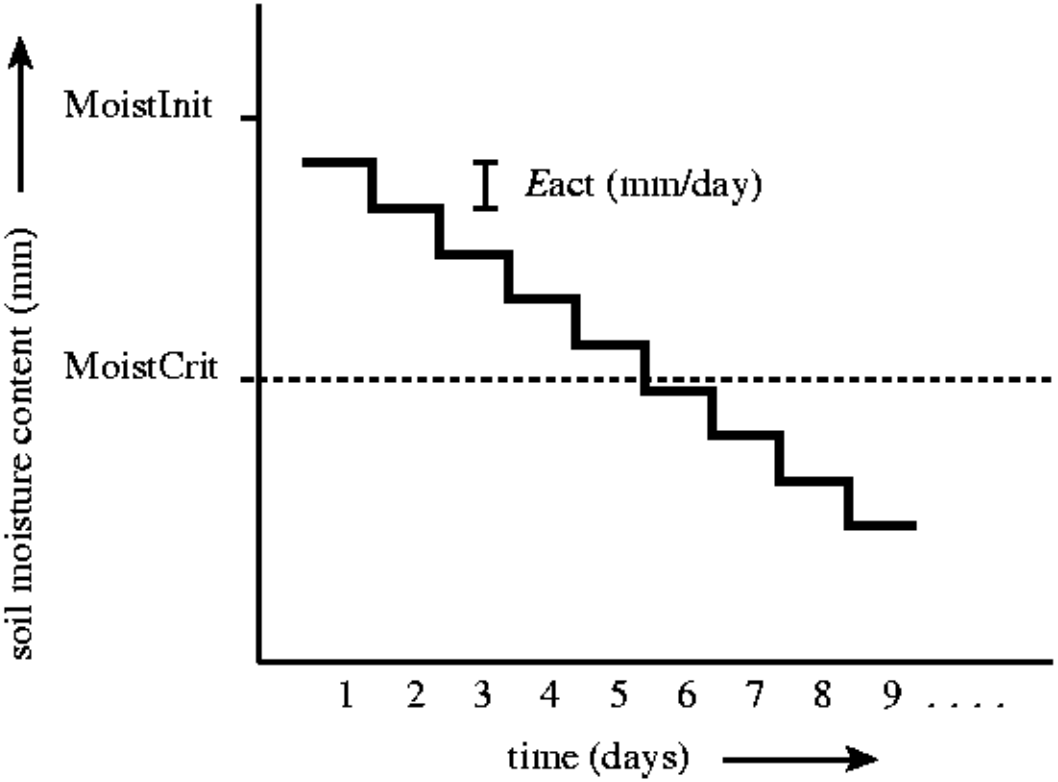
where *E*ref (mm/day) is the reference crop evapotranspiration of a specified green grass surface with a soil not short of water. *E*ref depends on the weather conditions during the day; for the sake of simplicity it is assumed to be constant. The soil coefficient *k*s (0 =< *k*s =< 1) mainly depends on the soil water content of the soil and is also assumed to be one: the soil is not short of water. The crop coefficient *k*c depends on the sort of crop and may have a value of 0.2 for almost bare ground up to 1.3 for vegetation that transpires at a great rate, such as corn. It is assumed that a map is available with the crop coefficient values, recoded from a land use map. So, the model assumes a temporarily constant evapotranspiration which for each day is given by:

*E*act =
*k*c x *E*ref (mm/day)

(2)

As a result of evapotranspiration the soil moisture content (mm) decreases. For simplicity, it is assumed here that evapotranspiration is the only flow that changes the soil moisture content. No other flows, such as infiltrating rain or percolation to the deeper groundwater occur. Figure 5-2 gives the change with time, with a timestep of 1 day, in the soil moisture content at a gridcell according to this concept. This Figure also shows the critical moisture content which is the moisture content value below which shortage in soil moisture may occur.

Figure 5-2. Change of Soil moisture content (mm) with time, timestep one day. Included process: evapotranspiration. MoistInit: moisture content at start of model run (mm); MoistCrit: critical moisture content (mm); *E*act: actual evapotranspiration (mm/day).



Now let's make a model for the process described above and shown in the Figure 5-2. Inputs for such a model are maps containing for each cell the initial moisture content and the critical moisture content. Additionally, in order to calculate the actual evapotranspiration a map with the crop coefficients for each cell and one constant value of the reference crop evapotranspiration is needed. The model must calculate (with equation (2)) and store in the database for each timestep a map with the moisture content at that timestep. Here we also assume that the model builder wants to know for each timestep the maximum moisture content in the study area and the area of land that has a moisture content below the critical value. Table 2 (below) gives the script for such a model. Two maps already present in the database are bound to the model names MoistMeas and Kc. Respectively, these are Moist952.map which contains measured moisture contents, which may be based upon an interpolation of field measurements, and CrCoef95.map which contains the crop coefficient value for each cell. The MoistCrit is the critical moisture content which is assumed to have a constant value of 20 mm over the study area. The reference crop evapotranspiration (modelname Eref) has a constant value of 8 mm/day. The binding also gives the file name under which the output time series TimeSeriesMin and TimeSeriesMax (defined in the dynamic section) is stored to the database.

Example 5-2. Dynamic modelling script for change in soil moisture content; included process evapotranspiration.

```
# model for calculation of reduction in soil moisture content
# incorporated processes: evapotranspiration
# timestep one day
```

```
binding
 MoistMeas = Moist952.map;   # measured moisture content (mm)
 MoistCrit = scalar(20);     # critical moisture content (mm)
 Eref = scalar(8);           # reference crop evapotranspiration (ETPrc)
                             # mm/day
 Kc = CrCoef95.map;          # crop coefficient map
 TimeSeriesMax = Max8.tss;   # time series binding
 TimeSeriesMin = Min8.tss;   # time series binding

areamap
 Clone.map;                  # clone map with location attributes of maps
                             # used in model

timer
 1 30 1;                     # starttime: 1 (first day)
                             # endtime: 30 (thirtiest day)
                             # timeslice 1 day
                             # as a result 30 timesteps (iterations)

initial
 report Eact = Kc * Eref;    # actual evapotranspiration (mm)
 Moist = MoistMeas;          # initial moisture content is measured
                             # moisture content (mm)

dynamic
 report Moist = Moist - Eact;              # moisture content (mm)
 MoistBelowCrit = Moist le MoistCrit;      # results in boolean map
 report TimeSeriesMax = mapmaximum(Moist);
 report TimeSeriesMin = mapminimum(Moist);
                                 # reports to a time series
```

All maps used in the model must have the same location attributes. The areamap section defines a map with these location attributes. Here it is the map Clone.map, available under that name in the PCRaster database.

The timer sets the model time. The model starts at time = 1 and ends at time = 30, with a timestep of 1. As a result it consists of 30 timesteps which represent 30 days of evapotranspiration. The starttime is 1, so the results of running the dynamic section for the first time, at the first timestep, are stored with time extension 1.

The initial section defines the initial moisture contents at the start of the model run: these are assumed to be the measured moisture contents stored in the map MoistMeas. The actual evapotranspiration according to equation (2) is also defined. It does not change during the model run, for this reason it has been put in the initial section. It is also reported: the initial moisture content is stored to the database under the name Moist.

The dynamic section contains three **pcrcalc** operations. For each timestep, first the moisture content for that timestep is calculated, by subtracting the actual evapotranspiration from the moisture content of the previous timestep. For each timestep, the **report** keyword provides that the moisture content is stored to the database. At the end of the model run, the database will contain 30 moisture content maps. These have filename extensions referring to the time of the timestep at which each map has been generated. The resulting map of the first timestep at time = 1 is stored under the name Moist000.001, the map of the second timestep at time is 2 is stored as Moist000.002 etc. Remember that it would not be possible to report also the operation Moist = MoistMeas; in the initial section. That would result in a report of Moist which is made two times in one model which is not allowed.

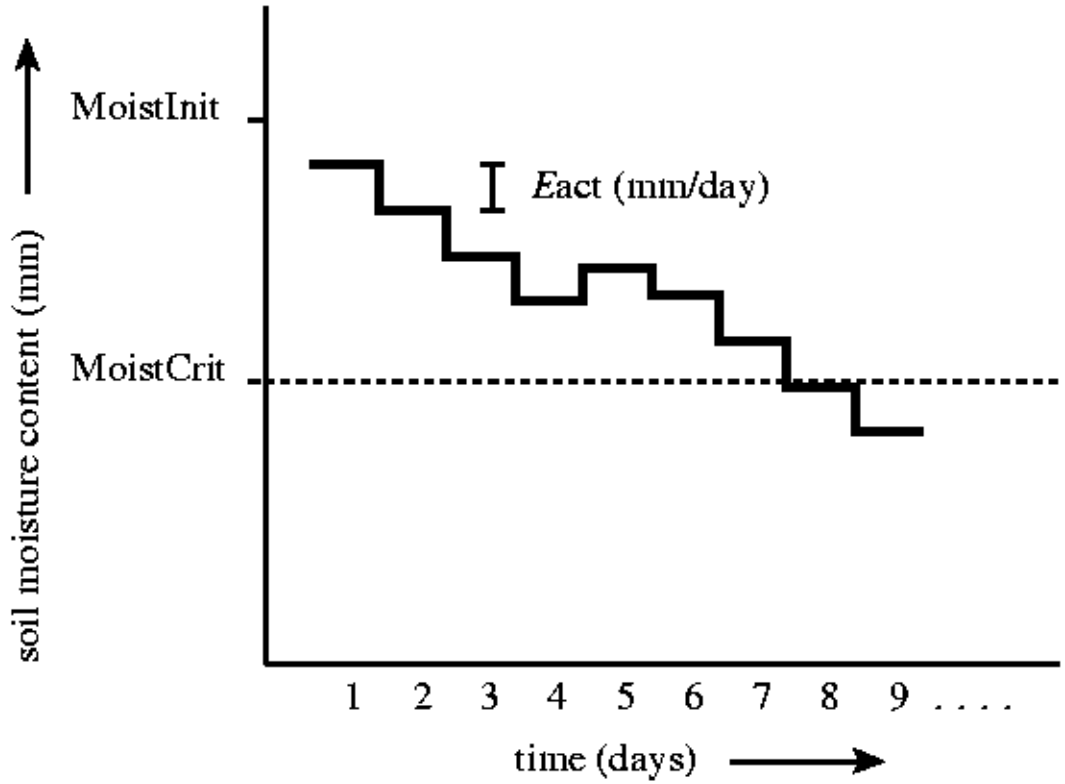
MoistBelowCrit is calculated in the second statement using the value for Moist that results from the **pcrcalc** operation in the first statement. MoistBelowCrit is a Boolean map that contains a Boolean TRUE (cell value 1) for cells that have a moisture content equal to or below the MoistCrit value and a Boolean FALSE for cells that still have a moisture content larger than MoistCrit.

The third statement reports a timeseries with model name TimeSeriesMax. Each timestep, the maximum cell value of the map Moist is written to the time series. The binding section defines that the table is stored under the file name Max8.tss. This is neatly done: for a next run with a slightly changed model (for instance with a Eact value of 6), the user only needs to bind the TimeSeriesMax to a new filename, for instance Max6.tss. In this way, it is relatively simple to run different scenarios, each time only changing the values and filenames in the binding.

5.1.4.2. Soil moisture model with timeinput: rain

Now, we build on the model given in the previous section by adding rain to the soil during the 30 days of evapotranspiration. We assume that all rain water immediately infiltrates in the soil. If the saturated soil moisture content is reached as a result of rain water infiltration, the remaining rain water is not added to the soil moisture anymore. This excess in rain will run off. Here, the runoff process is not incorporated in the model: it is assumed that the saturated moisture content is not exceeded as a result of infiltrating rain. The evapotranspiration rate is assumed not to be influenced by the infiltrating rain water. So no changes are made in the model with respect to evapotranspiration. Figure 5-3 shows the temporal change in soil moisture content as a result of evapotranspiration and infiltrating rain.

Figure 5-3. Change of Soil moisture content (mm) with time, timestep one day. Included process: evapotranspiration and rain. MoistInit: moisture content at start of model run (mm); MoistCrit: critical moisture content (mm); *E*act: actual evapotranspiration (mm/day). Rain falls at time = 6 and 7 days.



The model example below gives the sequential modelling script for evapotranspiration and infiltration. The binding section gives the model names for respectively the saturated soil moisture content map and the time series with the rain in millimetres for each timestep. The table below gives the ascii formatted time series file Rain.tss that is used.

Example 5-3. Dynamic modelling script. Included processes: evapotranspiration and infiltrating rain.

```
# model for calculation of reduction in soil moisture content
# incorporated processes: evapotranspiration and infiltration of rain
# timestep one day
```

```
binding
 MoistMeas = Moist952.map;   # measured moisture content (mm)
 MoistCrit = scalar(20);     # critical moisture content (mm)
 Eref = scalar(8);           # reference crop evapotranspiration;
                             # (ETPrc) mm/day
 Kc = CrCoef95.map;          # crop coefficient map
 TimeSeriesMax = Max8.tss;   # time series binding
 TimeSeriesMin = Min8.tss;   # time series binding
 RainTimeSeries = Rain.tss;  # timeseries with amount of rain at each
                             # timestep (mm/day)

areamap
 Clone.map;                  # clone map with location attributes of maps
                             # used in model

timer
 1 30 1;                     # starttime: 1 (first day)
                             # endtime: 30 (thirtiest day)
                             # timeslice 1 day; as a result
                             # 30 timesteps (iterations)

initial
 report Eact = Kc * Eref;    # actual evapotranspiration (mm)
 Moist = MoistMeas;          # initial moisture content is measured
                             # moisture content (mm)

dynamic
 Rain = timeinputscalar(RainTimeSeries, 1);
 report Moist = Moist + Rain - Eact;  # moisture content (mm)
 MoistBelowCrit = Moist le MoistCrit; # results in boolean map
 report TimeSeriesMax = mapmaximum(Moist);
 report TimeSeriesMin = mapminimum(Moist);
                             # reports to a time series
```

In the first statement of the dynamic section, the **timeinputscalar** operator reads the RainTimeSeries and assigns each timestep the amount of rain to Rain. In the second statement, Rain is added to the soil moisture Moist.

The remaining statements give the operations for the evapotranspiration process and the report. These have not been changed compared to the model given in the previous section.

Example 5-4. Time series Rain.tss: rain at the study site.

```
Rain, 1-30 July 1995
2
time
rain (mm)
1     0.0
2     0.0
3     5.2
4     0.0
5     23.1
```

```
6     40.1
.
.
- lines for 7 to 28 not shown -
.
.
29    0.0
30    0.0
```

Prev
Command syntax and script files for cartographic modelling

Home

Next
Functional list of PCRaster operators

Chapter 3. How to Import or Export Data, Display Maps, Global Options

Table of Contents

3.1. Introduction

3.2. PCRaster maps: database management

3.3. Tables: database management

3.4. Time series: database management

3.5. Point data column files: database management

3.6. Global options and local options

3.1. Introduction

This Chapter contains information about how to in- and export data from PCRaster. It discusses how to convert data from other GIS to one of the kinds of data used in PCRaster (Section 3.2.1) and how to export data from PCRaster (Section 3.2.2). Other database management operations are also described, such as how to cut or join together maps and how to change location attributes or the data type of maps (Section 3.2.3). Furthermore it will be explained how to attach a legend to a map (Section 3.2.4) and how to display PCRaster maps on the computer screen and print a map (Section 3.2.5). All operations are done using one of the PCRaster operators given in the text. For a detailed description of these operators, see the alphabetical list of operators in Reference I, *List of PCRaster Operators*.

The other sorts of data used in PCRaster (tables, time series and point data column files) are ascii formatted. Unlike PCRaster maps, these can easily be created, analyzed and edited with other software packages. Section 3.3 and Section 3.4 explain the creation of tables and time series, respectively. Creation and conversion of point data column files is described in Section 3.5. The above mentioned sections do not describe the formats of these sort of data. The format of PCRaster maps (Section 2.3), tables (Section 2.4), time series (Section 2.5) and point data column files (Section 2.6) are described succesively.

Section 3.6 gives a list of *global options* used in PCRaster. Global options are set only once (for instance at the beginning of a project) and will affect all operations to which they are relevant. *Local options* which are used each time a operation is done, must always be redefined.

Point data column files

PCRaster maps: database management

3.2. PCRaster maps: database management

3.2.1. Creation of a PCRaster map; data import

PCRaster maps are in a binary format which is only used in PCRaster. As a result you cannot analyze PCRaster maps with other software packages. So, it is important to know how to convert data (for instance point data column files) to and from PCRaster map format.

If you start a project and want to use PCRaster maps for analyzing your data, first make an empty clone map to define the geographical and cartographical location attributes and set this map as global clone map with the global option --clone. To make the clone map, use the operator **mapattr**; the location attributes can be entered using the menu given by the **mapattr**. This map can be used as a clone (mask map) for all data sets you want to import. A data type and its cell representation can also be attached to the clone map using **mapattr**. This data type and cell representation will by default be assigned to the data that are imported, but the type of the imported data can also be specified during the import.

The choice of geographical location attributes must be based upon the spatial characteristics of the data set you want to import to the PCRaster map format and upon whether x,y coordinates are attached to the data in the data set or not. There are two possibilities for data import:

1) import of point data with x,y coordinates using either a *column file in simplified Geo-EAS format* or *plain column file format*

The data file you want to import contains x and y coordinates with data values. In this case the location attributes of the PCRaster map must be chosen in correspondence with the spatial distribution of the data given by the x and y coordinates. If the data are regularly spaced on a rectangular grid, you probably want location attributes that match the set of x and y coordinates of the data set. If the data are irregularly spaced we advise to choose a map size of the smallest bounding rectangle (or somewhat larger) that comprises the study area, as shown in Figure 2-3. In this Figure a rectangle has been chosen that is rotated with respect to the real coordinate system (positive angle). Quite often, it is better not to rotate the map, and to choose an unrotated (smallest) rectangle. Rotation may result in a map that better fits the shape of the study area. But remember that rotation has an important effect which may be very clumsy: rotation will not only result in a rotated map but of course also in a grid of cells that is rotated with respect to the real coordinate system. As a result, cells that are in one row on the PCRaster map will not have the same y coordinates; the same holds for cells in one column and their x coordinates.

Two sorts of column files with x and y coordinates may be imported: a *column file in simplified Geo-EAS format* or a *plain column file format*. These formats are used in PCRaster for representation of point data, especially in the gstat module for geostatistical analysis. Section 2.6 describes these formats. Point data

are imported to PCRaster map format with the PCRaster operator **col2map**.

2) import of data without x and y coordinates (ascii formatted)

The data do not contain x and y coordinates: the asciifile with your data contains a sequence of cell values, without coordinates. In this case, the number of rows and columns of the PCRaster map must correspond exactly with the number of rows and columns of the file you want to import or else the import of data will result in nonsense. Among other data, maps from the ARC/INFO or Genamap GIS packages are imported in this way. It is done with the PCRaster operator **asc2map**.

3.2.2. Data export from a PCRaster map

Data can be exported from a PCRaster map using one of the operators **map2col** and **map2asc**.

The operator **map2col** exports data to an ascii column file in simplified Geo-EAS format or a plain column file format. Both contain x,y coordinates and data values. These kind of data are also used in the PCRaster package for representation of point data, see for formats Section 2.6. The plain column file format can easily be imported in spreadsheet, database management or word processing programs. It is also used in the gstat module of PCRaster.

The operator **map2asc** exports to an ascii file which will contain only data values, without x and y coordinates. This operator is used if you want to export data to the ARC/INFO package.

3.2.3. Cutting or joining maps; changing geographical location attributes or data types

The spatial characteristics of the spatial data in PCRaster map format can be changed with both the operator **resample** and **mapattr**. Note that these two operations have a totally different result:

If you want to resample your data in a PCRaster map to a new map with different geographical location attributes use the operator **resample**. First create a new map with the location attributes you wish (this is done with **mapattr**). For instance it may have geographical location attributes that define an area that only partly covers the old map, with a somewhat smaller or larger cell size. Now the **resample** operator can be used to resample your old data to the new map: for each cell of the new map a new cell value is computed on basis of the cell values on the old map that come into the cell on the new map.

The **resample** operator can also be used to join two maps together. The maps that are joined together will be resampled and may have different location attributes: for instance they may overlap or may not overlap or may have different cell sizes.

The geographical location attributes of a map can be changed using the operator **mapattr**. Using this operator will not result in resampling of the data: each cell of the new map will contain a value that corresponds with the value on that cell of the old map. For instance halving the cell width of a map that consists of 50 x 50 cells of width 10 m. results in a (smaller) map of 50 x 50 cells of width 5 m., containing values that are taken directly from the old map. So changing the location attributes with **mapattr** will result in a new location of your data with respect to the real world coordinate system.

Maybe this sounds silly but you may want to change the geographical location attributes after a map has been made, especially if you made an error in the specification of the location attributes with **mapattr**.

Conversion between data types is done using one of the conversion of data type operators (**boolean**, **nominal**, **ordinal**, **scalar**, **directional** and **ldd**. These operators change the data type of the input map to the data type that corresponds with the name of the operator. Conversion is only possible if it results in a map that has some meaning with the new data type attached to it.

3.2.4. Attaching a legend to a PCRaster map

The operator **legend** attaches a legend to a PCRaster map.

3.2.5. Screen display, hard copy output of PCRaster maps

hard copy output: not yet included in software

Screen display of PCRaster maps is done with the **display** operator. Time series can be plotted with the operator **timeplot**.

3.3. Tables: database management

3.3.1. Introduction

Section 3.3.2 describes how to create or edit a table.

3.3.2. Creating and editing tables

By default, the PCRaster package uses column tables. For defining relations between two maps of boolean, nominal, ordinal or ldd data type it is sometimes better to use matrices instead of column tables. If you want to use the matrix tables, set the global option --matrixtable. This option can be set for one separate operation or as general global option. How this is done, will be described later on in this chapter (Section 3.6). If a relationship is specified between more than two maps in the matrix, the relation can not be described by a matrix table: PCRaster will automatically use a column table.

A table for the PCRaster operator **lookup**, for creating a new map on basis of a table) or an input table for the operator **table**, for counting the number of cells that match the key) can be made with a text editor or alternatively with spreadsheet or word processing programs by exporting your table as a file in ascii text format. Additionally, the **table** operator itself generates a correctly formatted table, which can be used (possibly with a few edits) as input table for the **lookup** operator.

3.5. Point data column files: database management

3.5.1. Introduction

The format of point data column files has been explained in the previous chapter (Section 2.6). Underneath (Section 3.5.2) the database management with point data column filess described.

3.5.2. Creating point data column files, conversion to/from PCRaster maps

If you have spatial data in a spread sheet program, database management program or a package for statistics you can create a point data column file by exporting your data as ascii formatted text. Before exporting, we advise you to put the x and y coordinates in the first and second column respectively. The third and following columns may contain the data. After exporting as ascii text, you can check and edit the ascii column file with a text editor. If you want to convert it to a PCRaster map it must have the simplified Geo-EAS format or the plain column file format, which is described in the previous chapter (Section 2.6). Conversion is done with the **col2map** operator (Section 3.2.1) about importing to a PCRaster map or see the operator **col2map**. Interpolation of regular or irregular spaced point data to a PCRaster map can be done with the gstat module.

Additionally, you can create a point data column file by exporting data from a PCRaster map, with the **map2col** operator. The point data column file will be in plain column file format or in simplified Geo-EAS format. See the **map2col** operator.

3.6. Global options and local options

3.6.1. Introduction, setting global options

Options define the exact, detailed functionality of the PCRaster operators. Two types of options are used in PCRaster. *Local options* are set each time a PCRaster operation is performed. Local options only apply to one PCRaster operator and are specified in the command line directly after **pcrcalc** by typing -*localoption*, with one - character, where *localoption* is the local option you want to set for the operation.

Unlike local options, global options define general rules for a set of PCRaster operators. Normally, they are set only once (for instance at the beginning of a project) and affect all operations to which they are relevant. If you use the MSDOS version of PCRaster, you can set general global options by typing after the DOS prompt:

set PCROPTIONS =
--globaloption*1*
--globaloption*2*
...--globaloption*n*

where globaloption1 is one of the global options, which, unlike a local option, is preceded by *two* - characters. If you use the UNIX version of PCRaster, global options are set by typing after the UNIX prompt:

PCROPTIONS='--globaloption*1*
--globaloption*2*
...--globaloption*n*';

export PCROPTIONS

Note that in UNIX no spaces are typed on either side of the = sign. For instance:

```
 PCROPTIONS='--clone
CloneStudyArea.map --lddin'; export
PCROPTION
```

After applying PCROPTIONS the global options have the setting as specified or, if they are not specified, the default values. This set of options is used until a new set of options is specified with PCROPTIONS. If you set the options again with PCROPTIONS, options which are not specified that time are always set to default, *no* old settings are taken.

If you want to set a different global option for only one operation, you can specify a global option in the command line. This is done in the same way as it is done for a local option, by typing the --globaloption after **pcrcalc**. If a non-**pcrcalc** operator is applied (for instance **table**) the global option is typed after the operator. The general global option setting described above is overruled only for the execution of the

operation that is given in the command line.

You will find the local options and global options in the alphabetical list of PCRaster operators (Reference I, *List of PCRaster Operators*). In this list, the local and global options are given for each PCRaster operator. All global options are listed and described in the next section.

3.6.2. Overview of global options

global options related to location attributes:

--clone *CloneMap*

The CloneMap is a PCRaster map that must have the location attributes of the maps you want to use during a project. It may be an empty map made at the start of a project using the operator **mapattr**, see also Section 3.2.1. Alternatively you may specify as CloneMap an existing PCRraster map containing data.

--unittrue (default) or --unitcell

This option specifies the units used for the coordinates and sizes of the cells. It is of importance to the operations that make calculations with distances or areas in the map or to operations that import or export coordinates of cells. Default, with the option --unittrue, PCRaster uses true distances and the true coordinate system of the map. These are given during creation of your map or the clone map of your map with **mapattr** and Section 2.3.2 about the location attributes of a map. The cell length is defined by the real length of a cell. The x coordinates are real distance coordinates and increase from left to right, starting with the x coordinate at the left edge of your map; the y coordinates increase from top to bottom, starting with the y coordinate at the top edge of your map, or from bottom to top, starting with the y coordinate at the bottom edge of the map (depends on the projection you have chosen). PCRaster uses a sort of matrix coordinates if the option --unitcell is set. This option is seldom used. Both the real coordinate system and the real cell length of the maps (given with the **mapattr** operator) are totally ignored. In all operations a cell length of 1 is used. If coordinates are exported or imported, the top left corner of the map is assigned the x,y co-ordinates (0,0); x increases from left to right and y increases always from top to bottom. As a result the centre of the top left cell of a PCRaster map has always --unitcell x,y coordinates (0.5,0.5).

--coorcentre (default) or --coorul or --coorlr

This option gives the coordinate position that is linked to a cell. If coordinates of cells are exported, you can choose to export for each cell the x,y coordinates of the centre of the cell (--coorcentre, default), the upper left corner of the cell (--coorul) or the lower right corner of the cell (--coorlr). If point data with x,y coordinates are imported to a PCRaster map the option determines the assignment of point values that have x,y coordinates exactly at the edges of cells. This is only relevant to the operator **col2map**.

global options for defining the sort of directional data type:

--degrees (default) or --radians

Default, the program interprets directional data as degrees (domain [0,360>). If the option

--radians is set, directional data are interpreted and displayed as radians (domain [0,2pi>).

global options for defining cell representations:

These options define the representation of cell values used for storage and processing of data. The settings are only applied to the maps that are created: if you change the cell representation settings during a project, only the new maps you generate are assigned the new cell representation. The old maps created before you changed the settings will keep the 'old' cell representation.

scalar and directional data type: --single (default) or --double

Default, the cell representation is *single real* (--single). If you set --double it will result in *double real* representation of cell values. See also scalar data type, Section 2.3.3.5 and directional data type, Section 2.3.3.6.

nominal and ordinal data type : --small (default) / --large

Default, nominal and ordinal data are represented by *small integer* cell representation. If you set --large, *large integer* is used. See also and ordinal data type, Section 2.3.3.4.

global option for specifying the format of tables

--columntable (default) or --matrixtable

Default, the operators **lookup** and **table** use column tables (setting --columntable). If the option --matrixtable is set, a matrix table is used. For a description of these formats, see Section 2.4.

global options related to generation of a local drain direction map

--lddout (default) or --lddin

This option determines whether small catchments at the edge of the map are or are not considered as pits. See the operators **lddcreate** and **lddcreatedem**.

--lddfill (default) or --lddcut

This option determines the way the elevation model is modified in the core of pits. See the operator **lddcreatedem**.

global option for specifying the neighbourhood of cells

--diagonal (default) or --nondiagonal

This options specifies the neighbourhood of cells in the **clump** operation, see the **clump** operator.

global option for defining the sort of message printed during execution of an operation

--noprogress (default), --progress or --nothing

This option affects the message printed on the screen during execution of an operation. Default, the name, copyright and version of the software is printed and error messages if errors occur (--noprogress). The option --progress will result in additional information printed during execution, such as 'busy with row x', etc. If you set the option --nothing, nothing is printed on the screen, except error messages.

2.3. PCRaster maps

2.3.1. Introduction

This section describes the format of the main sort of data used in the PCRaster package: the PCRaster map, which contains spatial data in raster format. A header is attached to each PCRaster map; it contains both the location attributes and the data type of the map. The location attributes define the position of the map with respect to a real world coordinate system, the size and shape of the map and its resolution (cell size). The sort of attribute stored in the map is given by the data type of the map. The data type determines the PCRaster operations that can be performed on the map. Data typing used in PCRaster helps to structure your data.

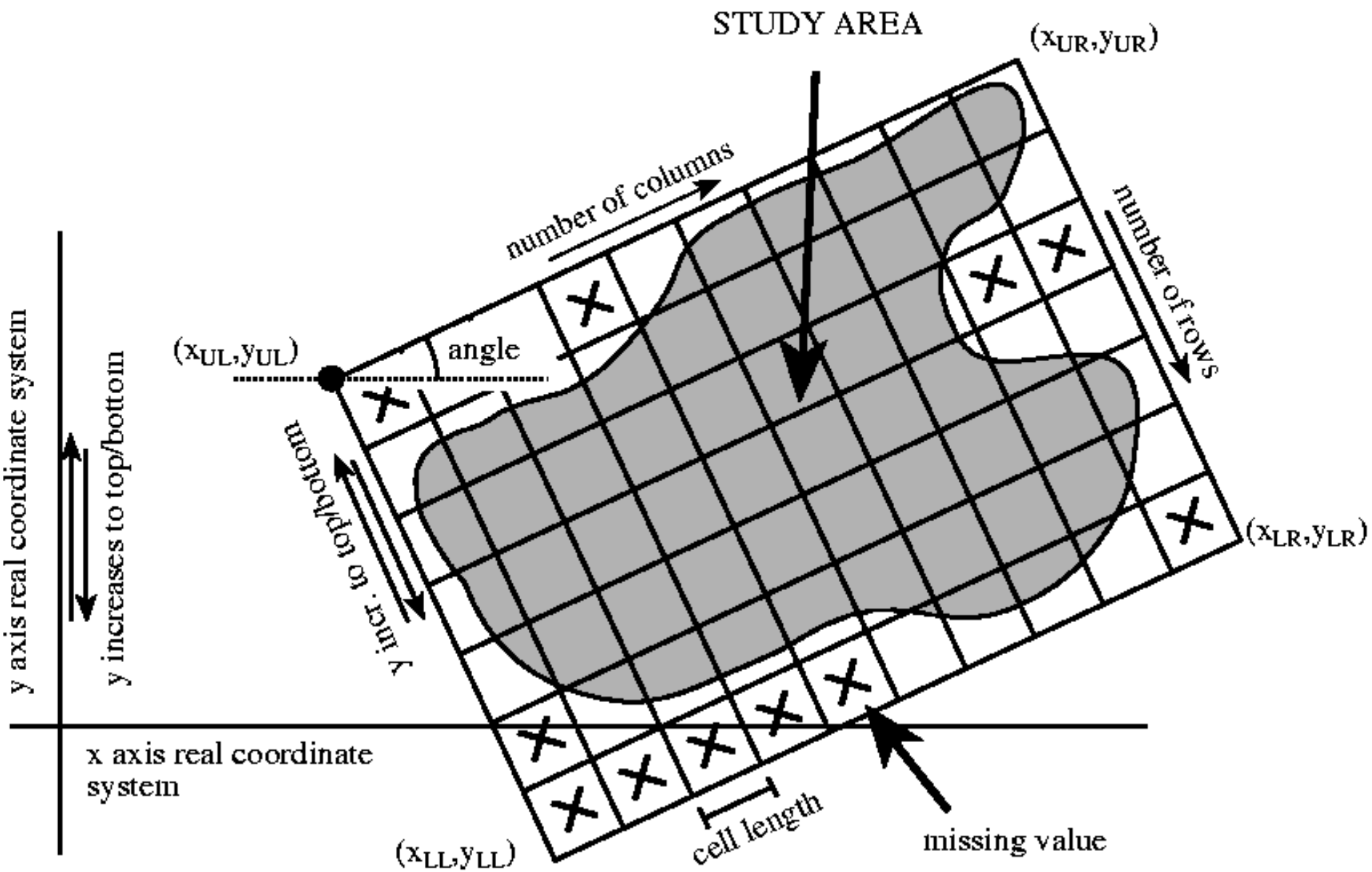
If you start a project, and want to import data to the PCRaster package in PCRaster map format it is wise first to make a map containing the header with the correct location attributes and the data type of the first data set you want to import. How this is done is described in Section 3.2. This section also describes other aspects of database management with a map.

2.3.2. Location attributes, missing values

This section gives an overview of the geographical location attributes linked to a PCRaster map.

The location attributes *projection*, x_{UL},y_{UL}, *cell length*, *number of rows*, *number of columns* and *angle* are used to define the position of the map with respect to a real world coordinate system and the shape and resolution of the map. Figure 2-3 shows schematically a PCRaster map of a study area and the location attributes used. As shown, the location attributes define the map with respect to the real world coordinate system (an ordinary x,y coordinate system).

Figure 2-3. Location attributes used to define the spatial characteristics of a PCRaster map. For explanation, see text.


STUDY AREA
(x_{UR},y_{UR})
number of columns
number of rows
(x_{UL},y_{UL})
angle
y incr. to top/bottom
(x_{LR},y_{LR})
y axis real coordinate system
y increases to top/bottom
x axis real coordinate system
(x_{LL},y_{LL})
cell length
missing value

The choice of the location attributes must be based upon the shape of the study area and the data set you want to store in the map. PCRaster maps always have a rectangular shape, but the shape and size of the map does not need to correspond exactly with the shape of the area studied, as shown in Figure 2-3: during data import to the PCRaster map the cells in the map outside the study area are assigned *missing values*. A missing valued cell is a cell which contains no attribute value. Missing valued cells are considered not to be included in the study area: PCRaster GIS and Cartographic or Dynamic Modelling operators ignore the missing valued cells. In general, cells that have a missing value on an input map of an operation are assigned a missing value on the resulting output map(s) also.

For a complete description of the choice of the location attributes related to the data set that will be stored in the map, see Section 3.2.1.

The location attributes have the following meaning; see also Figure 2-3:

projection

The projection of the real coordinate system which will also be assigned to the PCRaster map, is assumed to be a simple x,y field (also used in basic mathematics). The x coordinates increase from left to right. The y coordinates increase from top to bottom or from bottom to top. This can be chosen; from top to bottom is default.

x_{UL},y_{UL}

The x_{UL}, y_{UL} are the real world coordinates of the upper left corner of the PCRaster map. The location of the PCRaster map with respect to the real world coordinate system is given by this corner: if a rotated map is used (an angle not equal to zero), it is rotated around this point (so rotation over 90 degrees will result in a x_{UL}, y_{UL} that is at the bottom left side in Figure 2-3). Other PCRaster map corners are x_{LL}, y_{LL} ; x_{UR}, y_{UR} ; x_{LR} , y_{LR} .

cell length

The cell length is the length of the cells in horizontal and vertical direction. This implies that cells in a PCRaster map are all of the same size and always square. The cell length is measured in the distance unit of the real world coordinate system.

number of rows, number of columns

The number of rows and the number of columns are the number of rows and columns of the PCRaster map respectively. The cell length multiplied by the number of rows and number of columns is the height and width of the PCRaster map, respectively (in distance units of the real world coordinate system).

angle

The angle is the angle between the horizontal direction on the PCRaster map and the x axis of the real world coordinate system. It must be between -90 and 90 degrees; a map with a positive angle has been rotated counter clockwise with respect to the real coordinate system, a map with a negative angle has been rotated clockwise. In most cases an unrotated map will be sufficient (angle = 0 degrees), see also Section 3.2.1.

2.3.3. Data types

2.3.3.1. Introduction

Data stored in PCRaster maps can be grouped according to the sort of attribute they represent. For instance, a distinction is often made between attributes that are stored in maps as classified data (for instance soil classes) or continuous data (for instance elevation). In PCRaster, attribute information is linked to each map by specifying one of six data types. Each *data type* imposes a distinct domain of values that may occur on a map (whole values or fractional values, range of possible values) and whether some kind of order/scale is represented by the data (with or without order; linear or directional scale). If a legend is attached to a map, the map is subtyped by its legend: the attribute stored in the map is not only specified by the data type of the map, but also by the legend. Also the domain of a subtyped map is determined by the legend: it consists only of the map values linked to the classes given in the legend. As a result, PCRaster prevents some

operations that otherwise would combine maps with different legends. For instance a landuse map and a soil map cannot be joined laterally together. The legend of a map and the resulting *subtype* is described in Section 2.3.4.

The data type mechanism used in PCRaster will help you understand and organize your ideas about the attributes stored in your database or used in some kind of spatial model. The data types will prevent you from doing operations that are nonsense: each time a operation is done, the system checks the data type of the input maps and if the operations would result in nonsense an error message is given. Also, for some PCRaster operators the system adapts the way the operation is done to the data type of the input maps (this is called polymorphic behaviour of GIS operators). Additionally, the map resulting from an operation is given the data type that fits the sort of data that result from the operation.

Most data types have a distinct *cell representation* . The cell representation is not related to the *concept* of data type checking in the GIS, and for ordinary use it is of little importance: it only determines the way the values of the cells are stored and processed in the computer. The cell representations used in PCRaster are *single real* or *double real* for scalar and directional data and *small integer* or *large integer* for nominal and ordinal data. These are represented in the computer by REAL4 (single real), REAL8 (double real), UINT1 (small integer) and INT4 (large integer). UINT1, REAL4, REAL8, UINT1 and INT4 are also applied in other software, see for an exact description a standard book about computers in your library. By default, PCRaster automatically chooses the cell representation for each data type, so for ordinary use you do not need to take care of the cell representation. In some cases, especially if you want to store extremely large or small data values at a high precision you may want to choose a cell representation another than the default. This can be done with global options for defining cell representations. The cell representations for each data type are given in the next sections.

The data types are described in Section 2.3.3.2 up to and including Section 2.3.3.7; a short overview of the types is given in Table Table 2-1. Operators for creating and conversion between data types are listed in Section 3.2.3.

Table 2-1. List of data types, domaines for default cell representation, without legends

data type	description attributes	domain	example
boolean	boolean	0 (false), 1 (true)	suitable/unsuitable, visible/non visible
nominal	classified, no order	0...255, whole values	soil classes, administrative regions
ordinal	classified, order	0...255, whole values	succession stages, income groups
scalar	continuous, lineair	- 10exp(37)...10exp(37), real values	elevation, temperature
directional	continuous, directional	0 to 2 pi (radians), or to 360 (degrees), and -1 (no direction), real values	aspect
ldd	local drain direction to neighbour cell	1...9 (codes of drain directions)	drainage networks, wind directions

2.3.3.2. Boolean data type

The domain of the Boolean data type is 1 (Boolean TRUE) and 0 (Boolean FALSE). It is used for all attributes that only may have a value TRUE or FALSE, for instance 'suitable or unsuitable for maize', or to specify cells that come into a class or do not come into a class, for instance cells with a watch-tower or cells

without a watch-tower. A legend can be made for a map of data type Boolean; it has no effect on the domain of the map.

2.3.3.3. Nominal data type

The nominal data type is used for classified data without order. It represents attributes described by classes, for instance a map with soil classes. If the default cell representation is used (small integer), the domain consists of whole values equal to or between 0 and 255, so 256 different classes can be distinguished. Of course any number in the domain can be chosen to represent a class, but normally for each class a number is chosen starting with 1, and a value of 0 is chosen for cells that do not belong to a class. If the cell representation large integer is chosen (optional) the domain consists of all whole values between -2^{31} and 2^{31} and many more classes can be stored in the map.

A legend can be attached to a map of nominal data type, see Section 2.3.4. This results in subtyping of the map.

2.3.3.4. Ordinal data type

The ordinal data type is used for classified data that represent some kind of order. For instance stages of succession or soil texture measured at an ordinal scale (silt, sand, gravel for instance). If the default cell representation is used (small integer), the domain consists of whole values equal to or between 0 and 255, so 256 ordinal classes can be distinguished. Of course any number in the domain can be chosen to represent an ordinal class, but normally for the first class an ordinal value of one is chosen and for the next classes the values 1, 2, 3,.. etc.; a value of 0 is chosen for cells that do not come into a class. If the cell representation large integer is chosen (optional) the domain consists of all whole values between -2^{31} and 2^{31} and much more ordinal classes can be distinguished.

A legend can be attached to a map of ordinal data type, see Section 2.3.4. This results in subtyping of the map.

2.3.3.5. Scalar data type

The scalar data type is used for continuous data that do not represent a direction, for instance number of inhabitants, air particle concentration, amount of rain, elevation, or wind speed. The default cell representation is single real, which allows for storing and processing real values of data between $-1*10^{37}$ and $1*10^{37}$, using a maximum of six decimals. Optionally the cell representation double real can be used, it allows for storing and processing real values between $-2*10^{308}$ and $2*10^{308}$ with a maximal number of fifteen decimals.

2.3.3.6. Directional data type

The directional data type is used for continuous data that represent a direction. The domain depends on the sort of directional data that is used: if the global option --degrees is set (for global options see Section 3.6), the domain consists of real values equal to 0 or between 0 and 360 degrees and the number -1 for cells without a direction (-1 and [0,360> which means that 360 is not in the domain). If the global option --radians is set the direction is given in radians, the domain is [0,$2pi$ > and the number -1 for cells without a direction. The value -1 is not a missing value: it represents a cell for which no direction can be given. For instance a cell in a flat terrain does not have an aspect; as a result it has the value -1 on a map with aspects. The direction in the map of a directional value 0 depends on the location attribute angle of the map, see Section 2.3.2 : a cell value of 0 points to the North of the map (the y direction of the real world coordinate system), the remaining values increase in clock wise direction. In most cases the top of the map will be the North (location attribute angle = 0 degrees). In these cases a directional value 0 is to the top of the map and 90 degrees (East) corresponds with a direction to the right side of the PCRaster map.

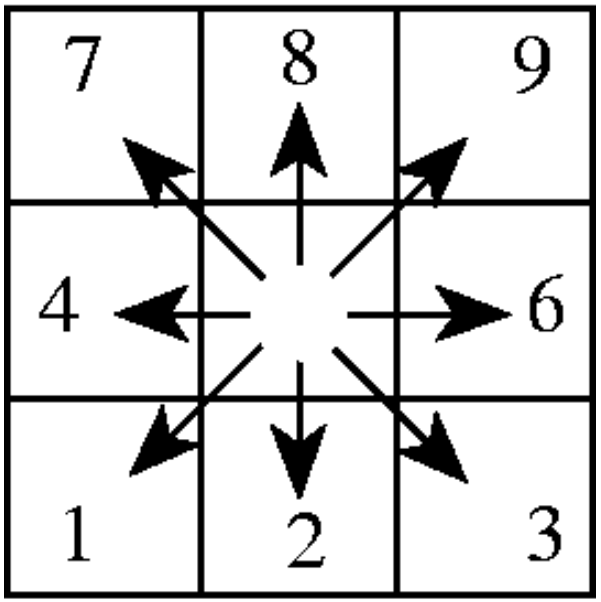
The directional data type can be used for all attributes that have a circular scale, for instance orientation or a year scale. Default the cell representation is single real; double real can be chosen for a higher precision, but in almost any case single real will give satisfying results. Note that statistics of directional data, like

mean and variance, are computed in a different way than for scalar data (see also [mardia72]). So always use the directional data type for directional data: PCRaster will automatically apply statistics for directional data to the map values.

2.3.3.7. Ldd data type

The ldd data type is used for maps that represent a local drain direction network . A local drain direction network is made up of a network of cells; each cell has a whole value from 1 to 9. These codes identify the neighbour of the cell to which material flows. The values have the meaning shown in Figure 2-4; note that the values are chosen to resemble the numeric key pad of your computer.

Figure 2-4. Directions of ldd codes. A value 5 (not shown) defines a cell without local drain direction (a pit).



For instance, during transport of material, a cell with value 3 designates flow to the bottom right neighbouring cell. The value 5 represents a **pit** : this is a cell without drainage to one of its neighbours.

Since the local drain direction network on a map of ldd data type defines a relationship between cells, a map of this data type must meet some requirements to safeguard these relationships. If a map meets these requirements it contains a so called *sound ldd network*. A ldd map is sound if it is a map containing only whole values from 1 to 9 or missing values. Additionally the values on the map must be ordered in such a way that each downstream path starting at a non-missing value cell ends in a pit cell. A downstream path consists of the consecutively neighbouring downstream cells; the pit cell at the end of the path is called the *outlet point* of the cell where the path started.

Here is a (non exhaustive) list of situations which cause a ldd to be *unsound*: - a cell on the border of the map has a local drain direction to the outside of the map. For example, a ldd code 7,8 or 9 on the first (top) row of cells or a value 7,4 or 1 on the first (left) column of cells of the map.
- a cell with a local drain direction to a cell with a missing value. For example a cell with a value 3 while its bottom right neighbour is a missing value.
- the ldd contains a cycle. A cycle is a set of cells that do not drain to a pit because they drain to each other in a closed cycle. The smallest cycle consists of two cells with local drain directions to each other; larger cycles may consist of several cells.

A ldd that is not sound cannot be used for PCRaster operations. So you must always prevent operations that may generate an unsound ldd. Normally, a ldd

network is made from an elevation map using the operator **lddcreate**. This will always result in a ldd that is sound. Other operations that can be used to generate a map of ldd data type will almost always result in a ldd that is unsound; examples are **asc2map**, **col2map**, **cover**, **lookup**. Some operations for making changes in a ldd must be done with care: editing using **display** and also cutting in a ldd map: always use the operator **lddmask** for cutting instead of for instance **if then**, **if then else**. A ldd that is not sound can be made sound using the operator **lddrepair**. Always use this operator if you are not sure whether your ldd is sound; it will be repaired if it is unsound.

2.3.4. Legends

Legend labels can be attached to boolean, nominal and ordinal maps with the operator **legend**.

Prev	Home	Next
Concepts, kinds of data used in the database	Up	Tables

col2map

Name

col2map -- Converts from column file format to PCRaster map format

Synopsis

col2map [options] columnfile PCRresult

columnfile	asciifile
PCRresult	specified by data type option; if data type option is not set: data type of *PCRclone* spatial

Options

--clone *PCRclone*

PCRclone is taken as clonemap. If you have set a global clonemap as global option, you don't need to set **clone** in the command line: the clonemap you have set as global option is taken as clonemap. If you have not set a global clonemap or if you want to use a different clonemap than the global clonemap, you must specify the clonemap in the command line with the **clone** option.

--unittrue or --unitcell

--unittrue

coordinates in columnfile are interpreted as real distance (default)

--unitcell

coordinates in columnfile are interpreted as distance in number of cell lengths

-B, -N, -O, -S, -D and -L

This data type option specifies the type options which is assigned to PCRresult (respectively boolean, nominal, ordinal, scalar, directional, ldd). If the option is not set, PCRresult is assigned the data type of *PCRclone* or the global clone. The data in columnfile must be in the domain of the data type which is assigned to PCRresult. For description of these domains see the description of the different data types.

--single or --double and --small or --large

In most case, the default cell representation will be sufficient. If you want, you can specify the cell representations: Scalar and directional data types

--single

cell values are represented by single real cell representations (default, single, precision)

--double

cell values are represented by double real cell presentations (double precision)

Nominal and ordinal data types:

--small

cell values are represented by small integer cell representaiton (default)

--large

cell values are represented by large integer cell representation

if option -D is set; --degrees of --radians

--degrees

values on `columnfile` are interpreted as degrees (default)

--radians

values on `columnfile` are interpreted as radians

-m *nodatavalue*

nodatavalue is the value in *columnfile* which is converted to a missing value on `PCRresult`. It can be one ascii character (letters, figures, symbols) or a string of ascii charaters. For instance: -m -99.89 or -m j5w. Default, if this option is not set, 1e31 is recognized as a missing value.

-s *seperator*

By default, whitespace (one or more tabs, spaces) is recognized as separator between the values of a row in the `columnfile`. If the values are separated by a different separator, you can specify it with the option. The *separator* can be one of the ascii characters (always one). In that case, **col2map** recognizes the specified separator with or without whitespace as separator. For instance, if the values in `columnfile` are separated by a ; character followed by 5 spaces, specify -s ; in the command line (you do not need to specify the whitespace characters).

columnnumbers

-x *columnnumberx*

is the column number of the x coordinate in `columnfile` (default 1)

-y *columnnumbery*

columnnumbery is the column number of the y coordinate in `columnfile` (default 2)

-v *columnnumberv*

columnnumberv is the column number of the cell values in `columnfile` (default 3)

Each cell on `PCRresult` is assigned the cell value on `columnfile` which has x,y coordinates that define a point in that cell";" for assignment of values in `columnfile` which have x,y coordinates at the edges of cells on `PCRresult`, the following options are used:

--coorcentre, --coorul or --coorlr

--coorcentre (default) or --coorul

values in `columnfile` that have x,y coordinates at the upper and left margins of a cell come into that cell, values at the bottom and right margins come into neighbouring cells. So, cell values with x, y coordinates at vertexes of cells come into the cell at the lower right side of the vertex.

--coorlr

values in `columnfile` that have x, y coordinates at the bottom and right margins of a cell come into that cell, values at the upper and left margins come into neighbouring cells. So, cell values with x, y coordinates at vertexes of cells come into the cell at the upper left side of the vertex.

Options to specify which value is assigned if two or more values in `columnfile` are found which all come into the same cell on `PCRresult`:

-a, -h, -l, -H, -M

-a

average value of the values found within the cell is assigned (default for scalar and directional data; for directional data and assignment of records without a direction, see notes)

-h

highest score: most occuring value found for the cell is assigned; if two values are found the same (largest) number of times, the highest value of these values is assigned, this is called a majority conflict (default for boolean, nominal, ordinal and ldd data)

-l

lowest score: least occurring value found for the cell is assigned (option for nominal, ordinal, boolean, ldd data); if two values are found the same (smallest) number of times, the smallest value of these values is assigned, this is called a minority conflict.

-H

highest value found for the cell is assigned (option for scalar or ordinal data)

-M

lowest value found for the cell is assigned (option for scalar or ordinal data)

Operation

The `columnfile` is converted to `PCRresult`, which is an expression in PCRaster map format with the location attributes of *PCRclone*. The `columnfile` must be in ascii format. Two types of column format can be converted; the **col2map** operator detects these formats by itself:

a column file in simplified Geo-EAS format:

line 1: header, description
line 2: header, number (*n*) of columns in the file
line 3 up to and including line *n* + 2: header, the names of the
n variables subsequent lines: data; they are formatted in at least three columns containing the x coordinates, y coordinates and values, respectively. Each line contains a record. The default column separator of the **col2map** operator is chosen to resemble the separator of the simplified Geo-EAS format.

a plain column file:

This is a file formatted like the simplified Geo-EAS format, but without header. The column separator may be different and can be specified with the option -s *separator*.

Fields with the x coordinates, y coordinates and values in the `columnfile` may contain the characters: -eE.0123456789. Fields may not be empty, valid fields are for instance:

-3324.4E-12 for -3324.4×10^{-12}
.22 for 0.22

For each cell on `PCRresult` the operator searches in `columnfile` for records that have x,y co-ordinates that come

into that cell on `PCRresult`. If one single record is found, the value of this record is assigned to the cell, if several records are found, the value which is assigned is specified by the option (-a, -h, -l, -H or -M). A cell on `PCRresult` without a value on `columnfile` that falls into the cell is assigned a missing value on `PCRresult`.

Notes

Directional data: If the option -a (average, default) is set, and both records without a direction (value -1) and records with a direction come into a cell (a so called direction conflict), the records without a direction are discarded and the cell value is computed from the records containing a direction only. Thus a cell is assigned a no direction value (value -1) only if all records for that cell don't have a direction. Using **col2map** for generating a `PCRresult` of data type ldd is quite risky: probably it will result in a ldd which is unsound (data types). If you do want to create a `PCRresult` of data type ldd use the operator **lddrepair** afterwards. This operator will modify the ldd in such a way that it will be sound, see the operator **lddrepair**.

Group

This operation belongs to the group of Creation of PCRaster maps

See Also

asc2map

Examples

1.

```
col2map --clone mapclone.map -S -m mv -v 4 ColFile1.txt Result1.map
```

```
ColFile1.txt
field data
5
xcoord
ycoord
pH1
pH2
code
25  25  mv  3.4  123
25  35  7.1  3.8  132
25  45  7.5  mv  123
35  25  7.4  3.1  123
35  35  7.7  3.0  321
35  45  7.8  3.9  123
45  25  7.1  3.4  321
45  35  7.4  3.1  213
45  45  7.8  3.7  321
55  25  7.4  3.2  314
55  35  7.1  3.2  141
55  45  7.9  3.7  132
mv  mv  7.4  3.2  111
mv  mv  7.6  3.1  111
```

Result1.map

7	5	10	MV
8	5	4	11
1	6	125	3

mapclone.map

1	1	1	1
1	1	1	1
1	1	1	1

2.

```
col2map --clone mapclone.map -O -m mv -x 2 -y 3 -v 6 --coorlr -H ColFile2.txt
```

Result2.map

ColFile2.txt

```
1  30  20  7.1  3.5  1  1
2  30  30  5.3  3.8  8  0
3  30  40  8.7  3.5  7  1
4  40  20  9.8  3.2  6  0
5  40  30  8.5  3.2  5  1
6  40  40  9.4  3.1  5  0
7  50  20  7.7  3.2  2  0
8  50  30  6.2  2.9  4  0
9  50  40  7.4  3.1  10  1
10  60  20  5.3  3.3  3  1
11  60  30  5.4  3.5  11  1
12  60  40  3.4  3.9  mv  0
13  1200  345  3.4  2.1  121  1
14  45.3  25.8  5.3  3.2  125  1
15  46.2  23.7  5.3  3.2  124  0
```

Result2.map

7	5	10	MV
8	5	4	11
1	6	125	3

mapclone.map

1	1	1	1
1	1	1	1
1	1	1	1

Prev asc2map | Home Up | Next legend

map2col

Name

map2col -- Converts from PCRaster map format to column file format

Synopsis

map2col [options] PCRmap1 PCRmap2....PCRmapn columnfile

PCRmap	boolean, nominal, ordinal, scalar, directional, ldd spatial The maps must have the same projection, the other location attributes and the data types may be different between the maps.
columnfile	asciifile

Options

--unittrue or --unitcell

--unittrue

coordinates in columnfile are interpreted as real distance (default)

--unitcell

coordinates in columnfile are interpreted as distance in number of cell lengths

coordinate positions:

--coorcentre

cell values on columnfile are assigned the coordinates of the centres of the cells on PCRmap1 (default).

--coorul

cell values on columnfile are assigned the coordinates of the upper left corner of the cells on PCRmap1.

--coorlr

cell values on columnfile are assigned the coordinates of the lower right corner of the cells on PCRmap1.

columnnumbers

-x *columnnumberx*

columnnumberx is the column number of the x coordinate in `columnfile`; in append mode: the columnnumber of the x coordinate in the *inputcolumnfile* (default 1).

-y *columnnumbery*

columnnumbery is the column number of the y coordinate in `columnfile`; in append mode: the columnnumber of the y coordinate in the *inputcolumnfile* (default 2).

-m *nodatavalue*

nodatavalue is the value in *columnfile* which is converted to a missing value on `PCRresult`. It can be one ascii character (letters, figures, symbols) or a string of ascii charaters. For instance: -m -99.98 or -m ! or -m j5w. Default, if this option is not set, 1e31 is recognized as a missing value.

-M

Default, if `PCRmap`1 has a missing value in a cell, the cell is not saved in `columnfile`. If the option -M is set these cells *are* saved in `columnfile`. They are assigned the *nodatavalue*.

-s *seperator*

By default, whitespace (one or more tabs, spaces) is recognized as separator between the values of a row in the `columnfile`. If the values are separated by a different separator, you can specify it with the option. The *separator* can be one of the ascii characters (always one). In that case, **col2map** recognizes the specified separator with or without whitespace as separator. For instance, if the values in `columnfile` are separated by a ; character followed by 5 spaces, specify -s ; in the command line (you do not need to specify the whitespace characters).

specifying format of columnfile

-p

`columnfile` is in plain format without header (default)

-g

`columnfile` is in simplified Geo-EAS format.

row wise or column wise output

-r

row wise output (default). The cell values in the first row on `PCRmap`1, `PCRmap`2,...`PCRmap`n will be at the top of the `columnfile`, underneath the cells in the second row, etc.

-c

column wise output. The cell values in the first column on `PCRmap`1, `PCRmap`2,...`PCRmap`n will be at the top of the `columnfile`, underneath the cells in the second column, etc.

-a *inputcolumnfile*

append mode. The `inputcolumnfile` is the name of the column file to which the data are appended. The file with the appended data is saved as `columnfile`. See operation in append mode below.

-f *C-typeformat*

This option is used to specify the sort of format which is assigned to the cell values in `columnfile`. The format determines the maximal and minimal cell value which can be converted, the precision the data can be saved in `columnfile` and the number of positions used for each cell value. The default format that is used depends on the data type of `PCRmap`. For boolean, nominal and ordinal maps, containing only whole values, the smallest possible number of positions is used for each cell value field in `columnfile`, taking into account all cell values and the number of positions needed for the missing value. For instance, a nominal map with nominal values between 12 and 19 and a no data value -999 (given by the option -m *nodatavalue*) is written to `columnfile` using four positions for each cell value (resulting from the number of positions needed for the *nodatavalue*). For instance, the value 12 is printed as a cell value field made up of 2 spaces followed by 12.
Maps of scalar and directional data type are always printed in the *C-type format* 11.6g, also used in the C programming language: the precision the data are saved on `columnfile` corresponds with the precision they are available on `PCRmap`, with the restriction that the maximal number of significant figures which can be saved on `columnfile` is six per cell value. The maximum and minimum value which can be saved is 10^{99} and -10^{99} respectively; a notation with $base_{10}$ exponents is used if the value is larger than 10^6 or smaller than -10^6.

Examples:

PCRmap	columnfile
-894.41000	-894.41
-5674935	-5.67494e+06
453628190.6	4.53628e+08
0.000000000031	3.1e-11
-0.02000012	-0.0200001
-1.0200001	-1.02

If you want to prevent the usage of base 10 exponents for scalar or directional data use the C-type format f and specify -f *a.d*f, where *a* and *d* must be whole numbers equal to or larger than 0. The value *d* is the number of decimal figures which will be used for each cell value, the value *a* is the minimal total number of positions used for each value; if more positions are needed (large values), more positions are used.

Examples:

C-typeformat	PCRmap	columnfile
5.6f	1234.1981	1234.198100
5.3f	1234.1981	1234.198
5.0f	1234.1981	1234
3.1f	1234.1981	1234.2
15.6f	1234.1981	1234.1981
2.1f	1289128932.75	1289128932.7

You can also specify other C-type formats, see for description of these formats a C programming language standard work.

Operation

Default operation (no append mode)
Operation if PCRmap1, PCRmap2,...PCRmapn have corresponding location attributes:
In most cases only one PCRmap1 is given in the command line or several maps PCRmap1, PCRmap2,...PCRmapn are given which have the same location attributes. In these cases, the operation is performed as follows. The PCRaster expression(s) PCRmap1, PCRmap2,...PCRmapn are converted to columnfile, which is an ascii file in column format. Each line in this columnfile represents one cell on PCRmap1, PCRmanp2,...PCRmapn. The x and y coordinates of the cells will be in the column numbers specified by the options -x *columnnumberx* and -y *columnnumbery*. The cell values of PCRmap1 will be in the first 'empty' column, the values of PCRmap2 in the next column etc. For instance if you set -x 2 -y 3, values of PCRmap1 are written in the first column, values of PCRmap2 in the fourth, values of PCRmap3 in the fifth etc.

Operation if PCRmap1, PCRmap2,...PCRmapn have different location attributes:
If more than one PCRmap is given in the command line and the given maps have different location attributes (with the exception of the projection which must be the same) the operation is performed in a somewhat different way. Only the x, y coordinates of the first map PCRmap1 are printed in columnfile. Real world coordinates or cell coordinates are printed, as specified by the option --unittrue, --unitcell; the coordinate position that is printed is specified by the option --coorul, --coorlr, --coorcentre. The cell values are printed as follows: first, each x, y coordinate pair is supplemented with its cell value of PCRmap1. Than, each line in columnfile is supplemented with the cell values of the remaining maps PCRmap2,...PCRmapn. No new lines are appended for these maps. For each of these maps and each line, the cell value is printed of the cell which has a real world location that corresponds with the real world location of the PCRmap1 cell that is already printed on that line. The real world location corresponds if the real world x,y coordinate of the PCRmap1 cell comes into the cell of the PCRmap2,...PCRmapn under consideration (the x,y coordinate of upper left corner, lower right corner or centre of each PCRmap1 cell is used, as specified by the --coorcentre, --coorlr, --coorcentre option). A line on the columnfile that represents a PCRmap1 cell with a real world x,y coordinate that does not come into a cell on the PCRmap2,...PCRmapn under consideration is assigned a missing value in the appended field.

operation in append mode
Data can also be appended to an existing column file *inputcolumnfile*. This columnfile may be a plain column file without a header or a column file in simplified Geo-EAS format. These formats don't need to be specified, the **map2col** operator will detect the format of inputcolumnfile itself. For the append mode, the option -a *inputcolumnfile* is used. The *inputcolumnfile* is the name of the column file to which the data are appended. The file with the appended data is saved as columnfile. The data are appended as follows: each line (record) of the *inputcolumnfile* is supplemented with the PCRmap1, PCRmap2,...PCRmapn values of the cell in which the x,y coordinates of the line (record) are. On each line, the values of PCRmap1, PCRmap2,...PCRmapn will be typed in the order they are specified in the command line (i.e. PCRmap1 values are printed in the first column after the columns in the *inputfile*, PCRmap2 in the second column, etc.).

The append mode results in the appending of columns only, no lines will be appended: a cell on PCRmap1, PCRmap2,...PCRmapn without a x,y coordinate in the *inputcolumnfile* that comes into the cell will *not* be saved in a new line (record). A line (record) on *inputcolumnfile* with a x,y coordinate that does not come into a cell on PCRmap1, PCRmap2,...PCRmapn is assigned a missing value in the appended column(s).

In append mode the options -M, -r, -c, -p and -g must not be used. The other options can be used as normal, but the default values will be appropriate in almost any case; the flags -m, -f and -s will only affect the columns

which are appended. The options --coorcentre, --coorul and --coorlr have the following meaning when used in append mode:

--coorcentre (default) and --coorul

lines (records) with coordinates in *inputcolumnfile* that are exactly at the upper or left edge of a cell are supplemented with the cell value of that cell, records with coordinates at the lower or right edge are supplemented with the value of a neighbouring cell.

--coorlr

lines (records) with coordinates in *inputcolumnfile* that are exactly at the lower or right edge of a cell are supplemented with the cell value of that cell, records with coordinates at the upper or left edge are supplemented with the value of a neighbouring cell.

Notes

Group

This operation belongs to the group of Creation of PCRaster maps

See Also

map2asc

Examples

1.

```
map2col --coorul PCRmap.map ColFile1.txt
```

ColFile1.txt

```
10          10      123.23
18          10           0
10          14        45.5
14          14         124
18          14       0.419
10          18       56.34
14          18          32
18          18       0.016
```

PCRmap.map

123	MV	0
45.5	124	0.419
56.3	32	0.016

2.

```
map2col -M -m -9999 -f 5.0f -g --coorcentre -c PCRmap.map ColFile2.txt
```

```
ColFile2.txt
map2col
3
x-coordinate
y-coordinate
PCRmap.map
    12    12   123
    12    16    46
    12    20    56
    16    12 -9999
    16    16   124
    16    20    32
    20    12     0
    20    16     0
    20    20     0
```

PCRmap.map

123	MV	0
45.5	124	0.419
56.3	32	0.016

3.

```
map2col --coorcentre -a ColFile2.txt PCRmap3.map ColFile3.txt
```

```
ColFile2.txt
map2col
4
x-coordinate
y-coordinate
PCRmap.map
map2col append: map2col.PCRmap3.imap
    12    12   123    5
    12    16    46    8
    12    20    56 1e31
    16    12 -9999    6
    16    16   124    9
    16    20    32 1e31
    20    12     0 1e31
    20    16     0 1e31
    20    20     0 1e31
```

PCRmap3.map

1	2	3
4	5	6
7	8	9

Prev map2asc | Home | Up | Next mapattr

4.7. Command syntax and script files for cartographic modelling

4.7.1. Introduction

PCRaster distinguishes two kind of operators. The first group of operators is meant for data management (including GIS functions); this is the group of data management operators in Chapter 6 . The second group, which is by far the largest, is meant for Map Algebra, Cartographic Modelling and Dynamic Modelling. These are the groups of operators for point operations, neighbourhood operations, area operations, map operations and time operations. Of these, the time operations are only meant for Dynamic Modelling, which will be discussed in the next chapter (Chapter 5). All the operations of the second group, except the **table** operator, use the keyword **pcrcalc** in the command line (the next section gives the exact syntax). **pcrcalc** is the program that does these operations. This program is not used for the operators for data management and the **table** operator.

Both operators that use **pcrcalc** and operators that do not use **pcrcalc** can be invoked from the command line by typing one operation after the DOS or UNIX prompt. The command syntax of this application of the operator will be explained later in this section (Section 4.7.2)

For Cartographic Modelling, the operations are combined in a script file. The operations in the script file are executed using the same **pcrcalc** program also used for separate **pcrcalc** operations. As a result only operations that use the **pcrcalc** program can be combined in a script file. Operations that do not use the **pcrcalc** program (data management operations, **display** and the **table** operator) are not used in a script file. Section 4.7.3 describes how to combine **pcrcalc** operations in a script file.

4.7.2. Command syntax

Both operations that use **pcrcalc** as well as operations that do not use **pcrcalc** can be invoked from the command line, by typing the operation after the DOS or UNIX prompt. This section first describes the operations that do not use calc, second the syntax of the operations that use **pcrcalc** is given.

Operations that do not use **pcrcalc** such as data base management operations and the **table** operation) are performed by typing after the prompt:

In DOS and UNIX:

operator [options] *InputFileName(s) ResultFileName(s)*

where operator is one of the non **pcrcalc** operators. Both the Inputfilename(s) and the Resultfilename(s) are filenames of one or more PCRaster maps, tables or point data column files. The kind of files depends

of course on the operator, see for a description per operator Reference I, *List of PCRaster Operators*. If more than one option is given, the options are separated by a space. For instance:

```
col2map -S -v 3 colfile.txt sodium.map
```

Operations that do use **pcrcalc** are given by typing after the prompt:

In DOS:

pcrcalc [options] *Result* = operator(*expression*1,...*expression*n)

In UNIX:

pcrcalc [options] '*Result* = operator(*expression*1,...*expression*n)'

where operator is one of the **pcrcalc** operators and *expression*1,...*expression*n are PCRaster maps or **pcrcalc** operations resulting in a PCRaster map with a data type that applies to the operator. In some cases a single number may be filled in for the expressions, see below. *Result* is a PCRaster map. Sometimes no brackets are used or the syntax is somewhat different. For the syntax per operator, see Reference I, *List of PCRaster Operators*. Whether quotes are used or not depends on whether the command line contains a DOS or UNIX special symbol . Without quotes a special symbol in the command is interpreted as having the meaning defined by DOS or UNIX. In UNIX the = sign is a special symbol. As a result quotes *must* be used in UNIX for all **pcrcalc** operations. No special symbols of DOS are used in **pcrcalc** operations except the '<' and '>' symbols in a few operations. So in general quotes are not needed in DOS. If the command contains a '<' '>' the quotes must be applied in the same way as it is (always) done in UNIX.

The use of **pcrcalc** operations for *expression*1,...*expression*n allows for nested operations . The number of operations used to define these expressions (so called because these may be expressions that result in a PCRaster map) is practically unlimited. For each separate statement, **pcrcalc** is typed only once, at the start of the line. For instance:

```
pcrcalc AspectTr.map = if(VegHeigh.map gt 5 then aspect(Dem.map))
```

uses the **if then**, **gt** and **aspect** operators (for a description, see Reference I, *List of PCRaster Operators*. Remember that the nested operations must result in correct data types . For instance, in the operation given above, the **gt** operation results in a map of boolean data type needed for the first expression of the **if then** operator. The second expression of the **if then** operator (typed after *then*) may have any data type, in this case it is directional.

Using a plain number (giving no data type specification, for instance the '5' in the operation given above) for an *expression* is possible if it is taken into account that the number that is filled in must be in the domain of the data type that is needed for the *expression* under consideration. For instance a value 8.7 cannot be filled in if a nominal data type is needed . In addition, a plain number may not be used if for determination of the data type of *Result* the data type of the *expression* for which the number is filled in is needed. For instance the following operation is *not correct*:

```
pcrcalc Friction.map = if(Boolean.map then 1 else 2)
```

because the data type of Friction.map cannot be determined on basis of '1' and '2'. In this kind of cases one of the data type assignment operators must be used; the correct operation for assigning a scalar data type to Friction.map is:

```
pcrcalc Friction.map = if(Boolean.map then scalar(1) else scalar(2))
```

4.7.3. Script files

4.7.3.1. Simple script files

Most geographic analyses contain a number of steps. PCRaster can be used to write down these steps in one script file and execute these steps sequentially. The thus created scripts are called Cartographic Models. A script contains a list of **pcrcalc** operations which describe the Cartographic Model. During execution of a script these operations are performed consecutively, from top to bottom in the script. Two layouts of a script can be used: with or without the *binding* section. First we describe how to use the plain script without the binding section. In the second part of this section we describe the script with binding section.

The script without binding has the following layout:

operation of calc;

operation of calc;

...

...

The script contains only **pcrcalc** operations, other operations which do not use the **pcrcalc** program (for instance **table**) cannot be used in a script. Each **pcrcalc** operation is typed on a separate line, omitting the word **pcrcalc** from the operation and terminated with a semi colon (;). On the right side of the = sign, an operation may use a file (a PCRaster map or for the **lookup** operation a table) that is present in the database, a file that is defined in the script or a plain number. The result (on the left side) of an operation is always a PCRaster map. Remark lines are preceded by a # character: everything typed on a line after this character is ignored and has no effect on the function of the model. An example of a script file is:

```
# this is a cartographic modelling script file
Friction.map = lookupscalar(Friction.tbl,LandUse.map);
Friction.map = 2.5 * Friction.map;
CostDist.map = spread(Start.map,0,Friction.map);
```

The first line is a remark, it is ignored by PCRaster. In the second line the script generates a map Friction.map with the **lookupscalar** operator from the files friction.tbl and LandUse.map, which must be present in the database. In the third line the Friction.map is multiplied by 2.5. In the fourth line CostDist.map is generated from the Friction.map defined in the script and Start.map already present in the database. In the fourth line the Friction.map is taken resulting from the last operation defining this

map, i.e. the values resulting from the operation in the third line.

Normally , without using **report** keywords, all resulting map values (on the left side of = sign) at the end of running a script are stored in the database, saving for each PCRaster map the last definition. In the example given above the Friction.map and the CostDist.map are stored in the database under these names. The last definition of Friction.map is stored, resulting from the third line. Alternatively specified results can be stored by typing the **report** keyword before a limited number of operations in the script. In that case, only the result of the operations preceded with **report** are stored. A certain resulting map name can be reported only once. For instance:

```
# example of a script file
Friction.map = lookupscalar(Friction.tbl,LandUse.map);
Friction.map = 2.5 * Friction.map;
report CostDist.map = spread(StartMap,0,Friction.map);
```

This script only stores CostDist.map in the database. If in addition the operation in the *second* line was also be preceded with **report**, then the values of the Friction.map resulting from the operation in this *second* line would be stored too. If the operation in the third line would be preceded with **report**, the values of the Friction.map resulting from the operation in this *third* line would be stored. Note that Friction.map may not be reported both in the second and the third line: this would result in reporting one map name twice in a script, which is not allowed.

The ascii formatted script can be created with a text editor or with a word processing program storing the file as ascii text. A script is executed by typing after the DOS or UNIX prompt:

pcrcalc -f *ScriptFileName*

where *ScriptFileName* is the name of the ascii formatted script file.

4.7.3.2. Script files with binding section

An extension to the plain script is the script with a binding section . In a plain script, files (PCRaster maps or tables) are directly linked to these in the PCRaster database: the names used in the script correspond with these in the database. A script with binding definitions allows for different names in the model part of the script. The binding binds the names of the files in the script to the names of the files in the PCRaster database. Often you may want to run a Cartographic Model script a number of times, each time with a different set of data files and with a different set of resulting files. In most cases, these data files are used a large number of times throughout the program. Using the binding, you only need to fill in the names of the files you want to use as input names and output names for the model run in the binding section, without changing all the file names in the rest of the script. Both file names used as input files for the model and names that are stored in the database during a model run with the **report** keyword may be given in the binding section. If the PCRaster DOS version is used, an additional advantage of the binding section is that it allows one to get rid of the DOS rules for filenames (format 8.3: xxxxxxxx.xxx) in the model part of the script: the unpractical DOS names with a dot can be replaced by model names which may be as long as you want, with or without a dot.

In the binding section, the name of a file in the database is bound (linked) to its name in the model with the following statement: binding is that it allows to get round the DOS rules for filenames (format

xxxxxxxx.xxx) in the script: the unpractical DOS names with a dot can be replaced by model names which may be as long as you want, with or without a dot.

A script with a binding definition is divided up in two sections: the binding section and the initial section. It has a structure as follows:

binding
binding statement;

...
...

initial

operation of calc;

operation of calc;

...
...

The binding section starts with the section keyword **binding**. Each line after this keyword contains one binding statement. Each statement gives a name for a PCRaster map or table in the script that is different from the file name of that variable in the database. Both file names used as input files for the model and names that are stored to the database during a model run with the **report** keyword may be given. The name of a file in the database is bound to its name in the model with the following statement:

```
NameInModel = databasefilename;
```

where databasefilename is the file name under which the variable is available in the database or will be stored to the database and NameInModel is the name used for the variable throughout the script.

Alternatively, a constant value can also be assigned to a PCRaster map variable. This applies only if ModelName is a PCRaster map which is an *input* to the model:

```
NameInModel = value;
```

where value is a number; NameInModel has this value throughout the Cartographic Model; its value cannot be changed. It has no data type attached to it. Attaching a data type to the PCRaster map NameInModel with a constant value is done by one of the data type assignment operators. We advice to specify the data type always because most operations need to know the data type of the maps used. An example that assigns a boolean data type to NameInModel is:

```
NameInModel = boolean(1);
```

The data type assignment operators are the only **pcrcalc** operations that can be given in the binding section, other operators cannot be used.

Not all variable names need to be defined in the binding section; as said above, the filenames of the variables in the database can also be used in the script.

The second section of a script with binding section is the initial section. It starts with the **initial** section keyword. The following lines contain the Cartographic Model, formatted in the same way as the plain script. Also the use of reports and # characters for remarks corresponds with the plain script. Using a binding, the plain script with a **report** might look like this:

Example 4-1. Example of a cartographic modellinge script file

```
# Example of a cartographic modelling
# script file with binding and initial section

binding
 # this is the binding section
 FrictionTable = Fr12.tbl;
 StartMap = Station.map;
 CostDistanceMap = CostRun1.map;

initial
 # this is the initial section
 FrictionMap = lookup(FrictionTable,LandUse);
 FrictionMap = 2.5 * FrictionMap;
 report CostDistanceMap = spread(StartMap,0,FrictionMap);
```

The initial section has not been changed. The binding section binds the Station.map, already present in the database, to the name in the model StartMap. The table Fr12.tbl in the database is used as FrictionTable. LandUse is not bound in the binding section; as a result it must be present in the database as a map with name LandUse. The report CostDistanceMap is stored in the database under the name CostRun1.map.

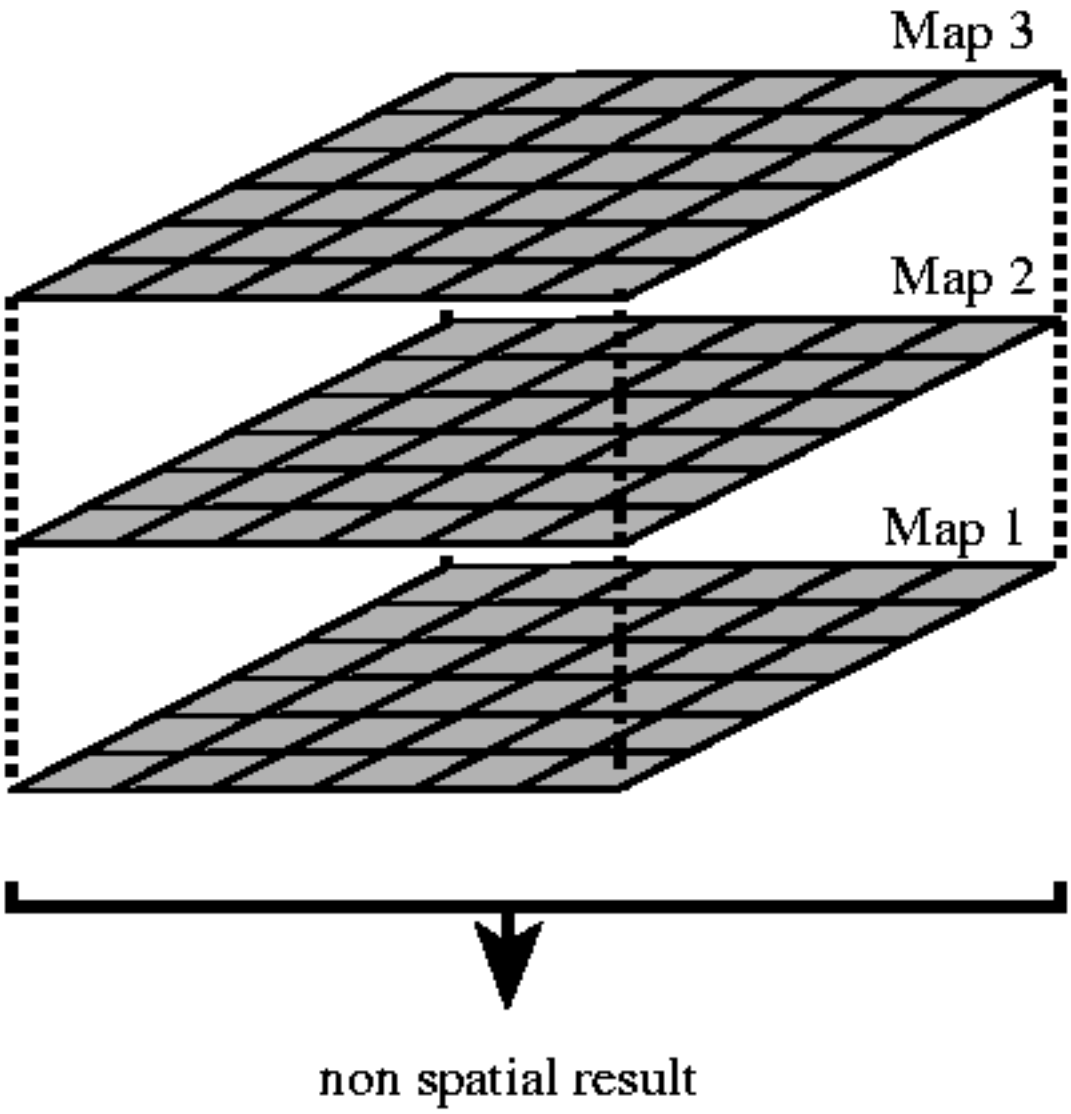
A script with binding section is also ascii formatted text. It is executed in the same way as the plain script.

Prev Map operations | Home | Up | Next Dynamic modelling

4.6. Map operations

The fourth group of PCRaster operations for Cartographic Modelling include those that compute one non-spatial value as a function of existing cell values of cells associated with a map, see Figure 4-9. The operations are like area operations to the extent that they compute a single value on basis of one or more map layers. Unlike area operations however, the value is determined on basis of all cells in a map.

Figure 4-9. Map operations. A non spatial value is calculated on basis of an aggregate value of a map or maps.



For a list of list of operations determine a statistical value on basis of all cell values in a map (for instance the maximum value) or result in a value related to the location attributes of a map, for instance the length of the cells on the map.

Prev Home Next
Area operations Up Command syntax and script files for cartographic modelling

4.5. Area operations

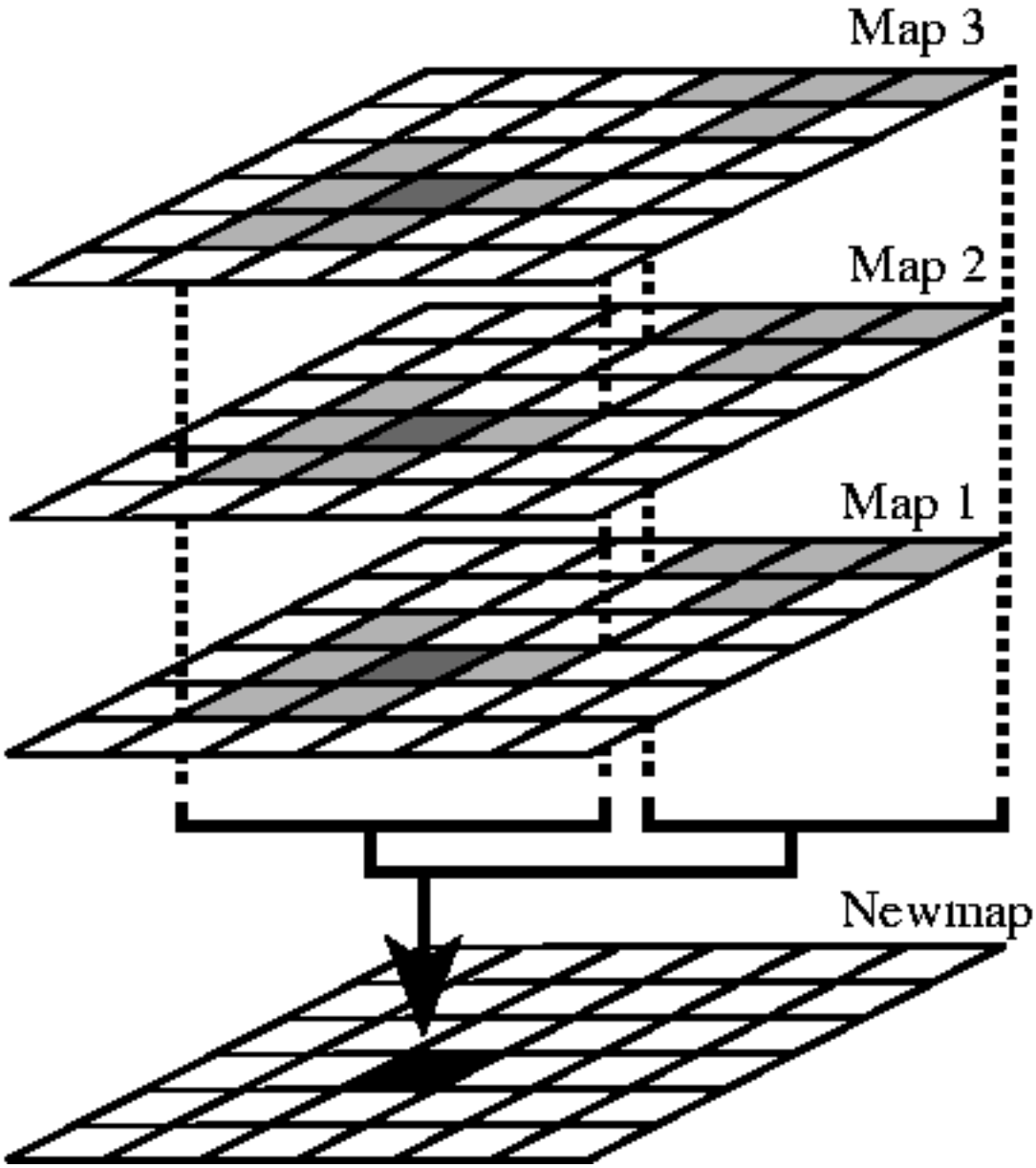
4.5.1. Introduction

The third group of PCRaster operations for Cartographic Modelling includes those that compute a new value for each cell as a function of existing cell values of cells associated with a zone containing that cell, see Figure 4-8. These operations provide for the aggregation of cell values over units of cartographic space (areas).

The operations are like point operations to the extent that they compute new cell values on basis of one or more map layers. Unlike point operations, however, each cell value is determined on the basis of the several cell values of cells in the zone containing the cell under consideration.

Area operations are also like neighbourhood operations to the extent that they represent operations for two-dimensional areas. But unlike neighbourhood operations, the constituent cells do not conform to any particular ordering or spatial configuration.

Figure 4-8. Area operations. The cell value is determined on basis of values of cells which are in the same area as the cell under consideration.



4.5.2. Operations over areas

The area operations use a nominal, ordinal or boolean map that contains the separate area classes. For each cel this map identifies the area class to which the cell belongs: cells with the same value on this map are member of a separate area class. These cells belonging to the same class do not need to be contiguous.

For each area class an operation is performed that calculates a statistical value on basis of the cell values of a second input map. This value is assigned to all cells on the resulting map belonging to the class. A wide range of operations is provided such as computation of area averages, determination of the minimum or maximum values within each area or calculation of the sum of the cell values within each area. Generation of a random number for each area is also possible.

6.4. Map operations

6.4.1. Operations over maps

mapmaximum
Determines the maximum cell value of all cells values.
mapminimum
Determines the minimum cell value of all cells values.
maptotal
Sums all cell values
maparea
Calculates the total area represented by the non missing value cells.
cellarea
Assigns the area of one cell.
celllength
Assigns the length which is identical in vertical and horizontal direction of one cell.

6.4.2. Random number generation - map

areanormal
Assigns to all cells one non spatial value taken from a normal distribution with a mean 0 and a standard deviation 1.
areauniform
Assigns to all cells one non spatial value taken from a uniform distribution with a mean 0 and a standard deviation 1.

Chapter 4. Cartographic Modelling

Table of Contents

4.1. Introduction

4.2. General approach to Cartographic Modelling

4.3. Point operations

4.4. Neighbourhood operations

4.5. Area operations

4.6. Map operations

4.7. Command syntax and script files for cartographic modelling

4.1. Introduction

This section describes the Cartographic Modelling part of the PCRaster package. As noted in Section 1.2 this module includes both analysis of maps using PCRaster operators for Map Algebra from the command line and using PCRaster operators for Cartographic Modelling. In Cartographic Modelling, PCRaster operators are used to build static models by combining several operations into script files. Using these script files, you can let the computer perform several PCRaster operations consecutively and automatically.

Cartographic Modelling does not include a concept of time: the operations that are performed represent one static change in the property of the cells. If you want to make models that incorporate processes over time (for instance hydrologic models using time series) you have to use the Dynamic Modelling module, which is described in Chapter 5. This module uses the same operators as used in Cartographic Modelling, but they are combined in a different way.

The first section of Chapter 4 describes the general approach to operations for Cartographic Modelling (Section 4.2). The subsequent sections describe the four classes of Cartographic Modelling operations: *point operations* (Section 4.3), *neighbourhood operations* (Section 4.4), *area operations* (Section 4.5) and *map operations* (Section 4.6). For an exhaustive list of all PCRaster operators included in these functional classes see Chapter 6. Section 4.7 describes the general rules for instructing your computer to perform PCRaster operations for Map Algebra and Cartographic Modelling (command syntax and script files).

If you have operated a geographical information system before and you have knowledge of Map Algebra, you may want to skip Section 4.2 up to and including Section 4.6 and use Chapter 6 to get an overview

of the operators. You should always read Section 4.7, because it contains important practical information.

Prev Home Next
Global options and local options General approach to Cartographic Modelling

Chapter 6. Functional list of PCRaster operators

Table of Contents

6.1. Point operators

6.2. Neighbourhood operators

6.3. Area operations

6.4. Map operations

6.5. Time operations

6.6. Data management

6.1. Point operators

6.1.1. Boolean operators

and
Performs a Boolean-AND operation on two expressions, on a cell-by-cell basis.
not
Performs a Boolean-NOT operation on two expressions, on a cell-by-cell basis.
or
Performs a Boolean-OR operation on two expressions, on a cell-by-cell basis.
xor
Performs a Boolean-XOR operation on two expressions, on a cell-by-cell basis.

6.1.2. Comparison operators

eq or ==
Performs a relational-equal-to operation on two expressions, on a cell-by-cell basis.
ge or >=
Performs a relational-greater-than-or-equal-to operation on two expressions, on a cell-by-cell basis.
gt or >
Performs a relational-greater-than operation on two expressions, on a cell-by-cell basis.
le or <=
Performs a relational-less-than-or-equal-to operation on two expressions, on a cell-by-cell basis.

lt or <
Performs a relational-less-than operation on two expressions, on a cell-by-cell basis.
ne or !=
Performs a relational-not-equal-to operation on two expressions, on a cell-by-cell basis.

6.1.3. Conditional statements

if then
For each cell a Boolean expression determines whether the value of an expression or a missing value is assigned to the result
if then else
For each cell a Boolean expression determines whether the value of the first expression or the value of a second expression is assigned to the result

6.1.4. Missing value creation, detection, alteration

cover
Substitutes missing values on an expression for values selected from one or more different expression(s), on a cell-by-cell basis.
defined
Assigns a Boolean TRUE for non missing values on the input expression and FALSE for missing values, on a cell-by-cell basis.
lddmask
Cuts a local drain direction map resulting in a (smaller) sound local drain direction map.
nodirection
For an expression of directional data type, returns TRUE for cells without a direction and FALSE otherwise for cells with a direction.
if then
For each cell a Boolean expression determines whether the value of an expression or a missing value is assigned to the result

6.1.5. Relations in tables

lookup...
For each cell, compares the cell value(s) of one or more expression(s) with the search key in a table and assigns a new value linked to that record in the key which matches the value(s) of the input expression
table
Creates on basis of one or more maps a table with a score for each key in the table. The score is the total area of the cells that match the key in the table.

6.1.6. Order

order
Returns ordinal numbers to cells in ascending order.
pred
For each cell returns an ordinal number which is the ordinal number of the next lower ordinal class (predecessor) on the expression.
succ
For each cell returns an ordinal number which is the ordinal number of the next higher ordinal class (predecessor) on the expression.

6.1.7. Maximize, minimize

maximize
For each cell, determines the maximum value of multiple expressions and assigns it to the corresponding cell for the result.
minimize
For each cell, determines the minimum value of multiple expressions and assigns it to the corresponding cell for the result.

6.1.8. Arithmic operators, trigonometric, exponential, logarithmic functions

*****
Multiplies the values of two expressions and sends this product to the result, on a cel-by-cell basis.
******
Calculates the *n*th power of the first expression, where *n* is the value on a second expression and sends it to the result, on a cell-by-cell basis.
-
Subtracts the value of the second expression from the value of the first expression and assigns it to the result, on a cell-by-cell basis.
+
Adds the values of two expressions and assigns this sum to the result, on a cell-by-cell basis.
/ of div
Divides the value of a first expressions by the value of a second expression and assigns this quotient to the result, on a cell-by-cell basis.
abs
Calculates the absolute value of an expression, on a cell-by-cell basis.
acos
Calculates the inverse cosine value of an expression, on a cell-by-cell basis.
asin
Calculates the inverse sine value of an expression, on a cell-by-cell basis.
atan

Calculates the inverse tangent value of an expression, on a cell-by-cell basis.

cos
Calculates the cosine of an expression, on a cell-by-cell basis.

exp
Calculates the $base_e$ exponential of an expression, on a cell-by-cell basis.

idiv
Divides (integer division) the values on a first expression by the values on a second expression and assigns this quotient to the result, on a cell-by-cell basis.

ln
Calculates the natural logarithm ($base_e$) exponential of an expression, on a cell-by-cell basis.

log10
Calculates the ($base_e$) logarithm of an expression, on a cell-by-cell basis.

mod
Divides (integer division) the values on a first expression by the values on a second expression and assigns the remainder to the result, on a cell-by-cell basis.

sin
Calculates the sine of an expression, on a cell-by-cell basis.

sqr
Calculates the square of an expression, on a cell-by-cell basis.

sqrt
Calculates the square root of an expression, on a cell-by-cell basis.

tan
Calculates the tangent of an expression, on a cell-by-cell basis.

6.1.9. Rounding

roundup
For each cell, the value of an expression is rounded upwards. Values of the results will be whole numbers.

rounddown
For each cell, the value of an expression is rounded downwards. Values of the results will be whole numbers.

roundoff
For each cell, the value of an expression is rounded off. Values of the results will be whole numbers.

6.1.10. Data types: Conversion and assignment

boolean
Converts from nominal, ordinal, scalar, directional or ldd data type to a boolean data type or generates a map of boolean data type with one constant value.

directional
Converts from boolean, nominal, ordinal, scalar or ldd data type to a directional data type or generates a map of directional data type with one constant value.

ldd

Converts from boolean, nominal, ordinal, scalar or directional data type to a ldd data type or generates a map of ldd data type with one constant value.

nominal

Converts from boolean, ordinal, scalar, directional or ldd data type to a nominal data type or generates a map of nominal data type with one constant value.

ordinal

Converts from boolean, nominal, scalar, directional or ldd data type to a ordinal data type or generates a map of ordinal data type with one constant value.

scalar

Converts from boolean, nominal, ordinal, directional or ldd data type to a scalar data type or generates a map of scalar data type with one constant value.

6.1.11. Random number generation - cells

normal

For each cell that is TRUE on a Boolean expression, assigns a value taken from a normal distribution with mean 0 and standard deviation 1.

uniform

For each cell that is TRUE on a Boolean expression, assigns a value taken from a uniform distribution between 0 and 1.

6.1.12. Coordinates, unique ID's

uniqueid

For each cell that is TRUE on a Boolean expression, assigns a unique whole value

xcoordinate

For each cell that is TRUE on a Boolean expression, assigns the xcoordinate of the cell

ycoordinate

For each cell that is TRUE on a Boolean expression, assigns the ycoordinate of the cell

---

Prev Home Next
Dynamic modelling Neighbourhood operators

6.2. Neighbourhood operators

6.2.1. Windows operations

windowaverage
For each cell, finds the average of cell values within a specified square neighbourhood and assigns it to the corresponding cell for the result
windowdiversity
For each cell, finds the number of unique values within a specified square neighbourhood and assigns it to the corresponding cell for the result
windowhighpass
Increases spatial frequency within a specified square neighbourhood. For each cell, it calculates the sum of cell values of an expression in a specified surrounding neighbourhood; this is subtracted from the cell values itself multiplied by twice the number of cells in the surrounding neighbourhood. The result of this calculation is assigned to the corresponding cell for the result.
windowmajority
For each cell, finds the most often occurring cell values within a specified square neighbourhood and assigns it to the corresponding cell for the result
windowmaximum
For each cell, finds the maximum cell value within a specified square neighbourhood and assigns it to the corresponding cell for the result
windowminimum
For each cell, finds the minimum cell value within a specified square neighbourhood and assigns it to the corresponding cell for the result
windowtotal
For each cell, finds the sum of cell values within a specified square neighbourhood and assigns it to the corresponding cell for the result

6.2.2. Derivatives of elevation maps

lddcreate
Creates a local drain direction map expression using the 8 points pour algorithm with flow directions from each cell to its steepest downslope neighbour. Pits can be removed with pit removing threshold map(s).
lddcreatedem
Creates a modified digital elevation model which fits the local drain direction map generated on the basis of the original digital elevation model (the elevation model is the input of the operation).
aspect
For each cell, calculates the aspect using elevations from a digital elevation model.

plancurv
For each cell, calculates the planform curvature (i.e. curvature transverse to the slope) using elevations from a digital elevation model.
profcurv
For each cell, calculates the profile curvature (i.e. curvature in the direction of the slope) using elevations from a digital elevation model.
slope
For each cell, calculates the slope using elevations from a digital elevation model.

6.2.3. Spread operations

spread
For each cell, calculates the friction-distance of the shortest material-distance path over a map with friction material from an identified source cell or cells to the cell under consideration.
spreadldd
For each cell, calculates the friction-distance of the shortest material-distance path over a map with friction material from an identified source cell or cells to the cell under consideration, where only paths are considered in downstream direction from the source cells.
spreadlddzone
For each cell, determines the shortest friction-distance path over a map with friction from an identified source cell or cells to the cell under consideration, where only paths are considered in downstream direction from the source cells. The value of the source cell at the start of this shortest material-distance path is assigned to the cell under consideration.
spreadzone
Determines for each cell the shortest friction-distance path over a map with friction material from an identified source cell or cells to the cell under consideration. The value of the source cell at the start of this shortest friction-distance is assigned to the cell under consideration.

6.2.4. Operations with local drain direction maps

ldddist
Calculates for each cell the material-distance of the path over a map with friction from the cell under consideration to the downstream nearest TRUE cell.
slopelength
For each cell assigns the accumulative-material-distance of the longest accumulative-material-path upstream over the local drain direction network to one of the cells against the divide of its catchment.

transport of material:
accuflux
For each cell calculates the accumulated amount of material that flows out of the cell into its neighbouring downstream cell. This accumulated amount is the amount of material in the cell itself plus the amount of material in upstream cells of the cell.
accucapacityflux and accucapacitystate
Transport input of material downstream over a local drain direction network; material is transported from one cell to its downstream cell when the transport capacity is exceeded, the remaining material is stored.

accufractionflux and accufractionstate
Transport input of material downstream over a local drain direction network; a fraction of the material is transported to its downstream cell, the remaining material is stored.
accuthresholdflux and accuthresholdstate
Transport input of material downstream over a local drain direction network; transport from one cell to its downstream cell only takes place if the amount of the material input to the cell exceeds the transport trigger of the cell: if the trigger is exceeded, all material is transported, else all material is stored.
accutriggerflux and accutriggerstate

miscellaneous operations:
catchment
Determines the catchment(s) (watershed, basin) of each one or more specified cells, subcatchments are not identified.
downstream
Returns the value of the neighbouring downstream cell.
downstreamdist
Returns the distance to the first cell downstream.
path
Determines for each TRUE cell on a Boolean input expression the path over the local drain direction network downstream to its pit; on the result, each cell which is on a path is assigned a TRUE.
pit
Assigns a Boolean TRUE to all pit cells on a local drain direction network.
streamorder
Assigns the stream order index to all cells on a local drain direction network.
subcatchment
Determines the (sub-)catchment(s) (watershed, basin) of each one or more specified cells, subcatchments are identified.
lddrepair
Repairs an unsound local drain direction map.
upstream
For each cell, assigns the sum of the cell values of its upstream cell(s).

6.2.5. Operations for visibility analysis

view
Assigns a TRUE or FALSE value for each cell on the result according to the visibility of that cell from one or more viewpoint cells in a 3D landscape defined by a digital elevation model.

Prev: Functional list of PCRaster operators
Home
Up
Next: Area operations

6.3. Area operations

6.3.1. Operations over areas

areaarea
For each cell, assigns the area of the area to which the cell belongs. Areas are identified by cell values on a expression with classes.
areaaverage
For each cell, assigns the average value of the cells that belong to the same area to the cell itself. Areas are identified by cell values on a expression with classes.
areadiversity
For each cell, assigns the number of unique cell values that belong to the same area to the cell itself. Areas are identified by cell values on a expression with classes.
areamajority
For each cell, assigns the most often occurring cell value of cells that belong to the same area to the cell itself. Areas are identified by cell values on a expression with classes.
areamaximum
For each cell, assigns the maximum value of the cells that belong to the same area to the cell itself. Areas are identified by cell values on a expression with classes.
areaminimum
For each cell, assigns the minimum value of the cells that belong to the same area to the cell itself. Areas are identified by cell values on a expression with classes.
areatotal
For each cell, assigns the sum of cells of cells that belong to the same area to the cell itself. Areas are identified by cell values on a expression with classes.
clump
Identifies all continuous groups of with the same value ('clumps'); cells belonging to one clump are assigned the same new unique value.

6.3.2. Random number generation - areas

areanormal
Assigns to each area one value taken from a normal distribution with a mean 0 and a standard deviation 1.
areauniform
Assigns to each area one value taken from a uniform distribution with a mean 0 and a standard deviation 1.

Neighbourhood operators Up Map operations

6.5. Time operations

6.5.1. Time operations

time
Assigns for each time step the time at that time step.
summary
not yet included in the software
For each stime step, writes the values of a given set of non spatial expressions to a time series.
timeinput
Assigns for each time step one of a set of maps in the database. Each time step, the map is taken with the extension that refers to the time at the time step.
timeinput...
For each time step assigns cell values read from a time series that is linked to a map with unique identifiers. Per time step, the time series gives for each unique identifier a cell value that is assigned to cells on the map with a corresponding unique identifier.
timeoutput
For each cell writes the expression value of an uniquely identified cell or cells to a time series. After a model run, the time series contains for each identified cell a list of expression cell values per time step.
timeslice
Assigns the timeslice.

6.6. Data management

6.6.1. Random number generation - areas

mapattr
Generates a new PCRaster map with attributes specified by the user or changes location attributes of an existing PCRaster map.

6.6.2. Conversion of data

asc2map
Converts from ascii file format (including ARC/INFO and GENAMAP ascii output) to PCRaster map format.
col2map
Converts from column file format (including simplified Geo-EAS format used in the GSTAT module of PCRaster) to PCRaster map format.
map2asc
Converts from PCRaster map format to ascii file format (including ascii input format for ARC/INFO.
map2col
Converts from PCRaster map format to column file format (including simplified Geo-EAS format also used in the GSTAT module of PCRaster).

6.6.3. Cutting and joining together PCRaster maps

resample
Cuts one PCRaster map or joins several PCRaster maps by resampling to the cells of the resulting PCRaster map.

6.6.4. Generation of legends

legend
Attaches a legend to or changes the legend of one or more maps.

6.6.5. Screen output

display
Displays one or more maps on the computer screen
timeplot
Plots one or more timeseries on a x,y lineplot on the computer screen.

6.6.6. Hard copy output

mapprint
not yet included in software

Prev Home Next
Time operations Up Glossary

+

Name

+ -- Addition

Synopsis

pcrcalc Result = expression1 **+** expression2

expression1	scalar
	spatial, non spatial
expression2	scalar
	spatial, non spatial
Result	scalar
	spatial; non spatial if expression1 and expression2 are non spatial

Operation

For each cell, the values of expression1 and expression2 are summed. This sum is assigned to the corresponding cell on Result.

Notes

A cell with missing value on expression1 and/or expression2 is assigned a missing value on Result.

Result += expression1 is an alternative notation for Result = Result + expression1

Group

This operation belongs to the group of Arithmetic operators

+

Examples

1.

```
pcrcalc  Result.map = Expr1.map + Expr2.map
```

Result.map

MV	8	2
2	MV	-6
100	-7	16

Expr1.map

2	6.2	-3
1	MV	7
86	-1	12

Expr2.map

MV	1.8	5
1	3	-13
14	-6	4

-

Name

- -- Subtraction

Synopsis

pcrcalc `Result = expression1 - expression2`

`expression1`	scalar
	spatial, non spatial
`expression2`	scalar
	spatial, non spatial
`Result`	scalar
	spatial; non spatial if `expression1` and `expression2` are non spatial

Operation

For each cell, the value of `expression2` is subtracted from the value of `expression1`. This difference is assigned to the corresponding cell on `Result`.

Notes

A cell with missing value on `expression1` or `expression2` or on both expressions is assigned a missing value on `Result`.

`Result -= expression1` is an alternative notation for `Result = Result - expression1`

Group

This operation belongs to the group of Arithmetic operators

-

Examples

1.

```
pcrcalc  Result.map = Expr1.map - Expr2.map
```

Result.map

MV	4.4	-8
0	MV	20
72	5	8

Expr1.map

2	6.2	-3
1	MV	7
86	-1	12

Expr2.map

MV	1.8	5
1	3	-13
14	-6	4

/ or div

Name

/ or div -- Division

Synopsis

pcrcalc `Result = expression1 / expression2`

pcrcalc`Result = expression1`**div**`expression2`

`expression1`	scalar
	spatial, non spatial
`expression2`	scalar
	spatial, non spatial
`Result`	scalar
	spatial; non spatial if `expression1` and `expression2` are non spatial

Operation

For each cell, the value on `expression1` is divided by the value on `expression2`. This quotient is assigned to the corresponding cell on `Result`.

Notes

A cell with 0 on `expression2` is assigned a missing value on `Result`. A cell with missing value on `expression1` and/or `expression2` is assigned a missing value on `Result`.

div is an alternative notation for /.

`Result` /= `expression1` is an alternative notation for `Result` = `Result` / `expression1`

Group

This operation belongs to the group of Arithmetic operators

See Also

idiv mod

Examples

1.

```
pcrcalc  Result.map = Expr1.map div Expr2.map
```

Result.map

MV	3.5	-3.5
0	3.2	-3.2
2.6	-2.6	8

Expr1.map

3.5	7	-7
0	9.6	9.6
5.2	5.2	56

Expr2.map

0	2	2
7	3	-3
2	-2	7

---

*

Name

* -- Multiplication

Synopsis

pcrcalc `Result = expression1 * expression2`

`expression1`	scalar
	spatial, non spatial
`expression2`	scalar
	spatial, non spatial
`Result`	scalar
	spatial; non spatial if `expression1` and `expression2` are non spatial

Operation

For each cell, the values of `expression1` and `expression2` are multiplied. This product is assigned to the corresponding cell on `Result`.

Notes

A cell with missing value on `expression1` and/or `expression2` is assigned a missing value on `Result`.

`Result *= expression1` is an alternative notation for `Result = Result * expression1`

Group

This operation belongs to the group of Arithmetic operators

*

Examples

1.

```
pcrcalc  Result.map = Expr1.map * Expr2.map
```

Result.map

0	-6	MV
23.5	56	0
0	-9	MV

Expr1.map

0	-2	MV
4.7	8	3
0	-6	0

Expr2.map

0	3	10
5	7	0
4	1.5	MV

Prev / or div
Home
Up
Next **

**

Name

** -- *n*th power of a first expression, where *n* is the value of a second expression

Synopsis

pcrcalc `Result = expression ** power`

`expression`	scalar
	spatial, non spatial
`power`	scalar
	spatial, non spatial
`Result`	scalar
	spatial if `expression` or `power` is spatial, else non spatial

Operation

For each cell, raises the cell values on `expression` to the *n*th power, where *n* is the cell value on `power`. The result of this calculation is assigned to the corresponding cell on `Result`.

Notes

A cell with a value 0 on `expression` and a value less than or equal to 0 on `power` is assigned a missing value on `Result`. Also, a cell with a value less than 0 on `expression` and a negative whole number on `power` is assigned a missing value on `Result`.

A cell with a missing value on `expression` and/or `power` is assigned a missing value on `Result`.

`Result **= power` is an alternative notation for `Result = Result ** power`

Group

This operation belongs to the group of Arithmetic operators

**

Examples

1.

```
pcrcalc  Result.map = Expr.map ** Power.map
```

Result.map

MV	1	0
128	MV	9
0.663	-64	0.111

Expr.map

0	2.5	0
2	-3	3
0.9	-4	3

Power.map

0	0	8
7	MV	2
3.9	3	-2

Prev Home Next
* Up abs

abs

Name

abs -- Absolute value

Synopsis

pcrcalc Result = **abs** (expression)

expression	scalar
	spatial, non spatial
Result	scalar
	dimension of expression

Operation

For each cell, calculates the absolute value of the expression cell value and assigns it to Result.

Notes

A cell with missing value on expression is assigned a missing value on Result.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.

```
pcrcalc  Result.map = abs(Expr.map)
```

Result.map

2	7	3.5
8.5	3.6	MV
0	14	0.8

Expr.map

2	-7	3.5
-8.5	3.6	MV
0	14	-0.8

Prev
**

Home
Up

Next
accucapacityflux, accucapacitystate

accucapacityflux, accucapacitystate

Name

accucapacityflux, accucapacitystate -- Transport of material downstream over a local drain direction network

Synopsis

```
pcrcalc Resultflux = accucapacityflux ( ldd, material, transportcapacity )
pcrcalcResultstate = accucapacitystate( ldd, material, transportcapacity )
pcrcalcResultflux, Resultstate = accucapacityflux, accucapacitystate (( ldd, material, transportcapacity )
```

`ldd`	ldd
	spatial
`material`	scalar
	spatial, non spatial
`transportcapacity`	scalar
	spatial, non spatial
`Resultflux`	scalar
	spatial
`Resultstate`	scalar
	spatial

Operation

These operations describe accumulation of material in a drainage network with a limited transport capacity of the drainage channel. For instance, when water is flowing through pipes or when sediment is transported whereby the transport is limited by the water velocity and thus transport capacity. The material that is not transported is stored. For each cell, **accucapacityflux** assigns the amount of material which is transported out of the cell, **accucapacitystate** assigns the amount which is stored in the cell. Both operators perform the same function of accumulation of material with a limited transport capacity, the only difference between the operators is the sort of result that is saved: **accucapacitystate** yields storages of material in cells, **accucapacityflux** yields fluxes of material out of cells.

For each cell, the amount of material input, for instance the amount of rain, is given by `material`. This is transported in downstream direction through the consecutively neighbouring downstream cells, following the local drain directions on `ldd`. Each time material moves through a cell a fraction is stored in the cell. These storages are saved as `Resultstate`, if the **accucapacitystate** operator is used. The remaining material is transported out of the cell, these amounts of outflow from each cell into its neighbouring downstream cell are the result of the **accucapacityflux** operator, they are saved as `Resultflux`.

The function can be described by flow of material through a set of linked systems, where a cell represents a system. The flow starts at the cells/systems at the watershed boundaries (defined by `ldd`) and ends at a pit cell. The systems are linked by the local drain directions on `ldd`, these define the path of flow through the set of cells/systems. Each time a system is passed, the amount of flow changes.

For a cell/system somewhere in the map, the flow of material is described by a system. The inflow of the cell is the sum of the outflow amounts of its upstream neighbours. This inflow amount is added to the `material` value in the cell itself. This amount of material is potentially available for transport out of the cell. The amount which is actually transported out of the cell is limited by the `transportcapacity` value of the cell: if the sum of material is larger than the `transportcapacity` value, the amount of material which is transported out of the cell is equal to the `transportcapacity` value, the remaining material is stored in the cell. If the sum of material is equal to or smaller than the `transportcapacity` value, all material is transported out of the cell and nothing will be stored in the cell.

For each cell, the amount of material which is transported to its downstream neighbour (or out of the map if the cell is a pit cell) is saved as `Resultflux` (use the operator **accucapacityflux**); the amount of material which is stored to the cell is saved as `Resultstate` (use **accucapacitystate**)

Notes

The values on `material` and `transportcapacity` must be equal to or larger than zero.

A cell with missing value on `material` and/or `transportcapacity` is assigned a missing value on `Resultflux` or `Resultstate`. Additionally, all its downstream cells are assigned a missing value.

The local drain direction network on `ldd` must be sound.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

Section 4.4.5 lddmask

Examples

1.

```
pcrcalc  Flux1.map, State1.map = accucapacityflux, accucapacitystate (Ldd.map, Material.map, 1.5)
```

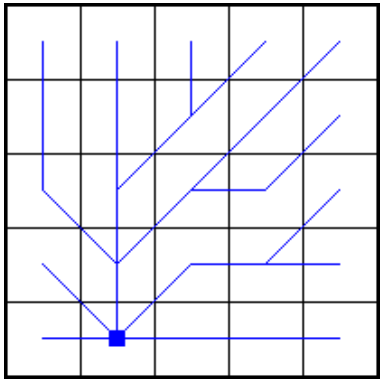
Flux1.map

1.5	0.5	1.5	1.5	1.5
1.5	1	1.5	1.5	1.5
1.5	1.5	1.5	1.5	0
0.5	1.5	1.5	0	0
0.5	1.5	1.5	1.5	1.5

State1.map

4.5	0	0.5	0.5	0.5
0.5	0	3.5	2	0.5
0.5	1.5	3.5	2	0
0	3.5	4.5	0	0
0	10	6	6	4.5

Ldd.map



Material.map

6	0.5	2	2	2
0.5	0.5	2	2	2
0.5	0.5	2	2	0
0.5	0.5	6	0	0
0.5	6	6	6	6

2.

```
pcrcalc  Flux2.map, State2.map = accucapacityflux, accucapacitystate (Ldd.map, Material.map, TransCap.map)
```

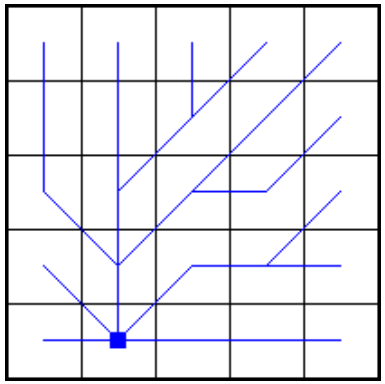
Flux2.map

10	2	10	10	10
20	2	30	20	10
30	2	30	20	10
10	2	30	30	30
30	2	30	30	30

State2.map

0	8	0	0	0
0	10	0	0	0
0	40	20	0	0
0	70	10	20	20
20	150	50	50	19

Ldd.map



Material.map

10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	50
50	50	50	50	49

TransCap.map

30	2	30	30	30
30	2	30	30	30
30	2	30	30	30
30	2	30	30	30
30	2	30	30	30

Prev
abs

Home
Up

Next
accuflux

accuflux

Name

accuflux -- Accumulated material flowing into downstream cell

Synopsis

```
pcrcalc Resultflux = accuflux ( ldd, material )
```

`ldd`	ldd
	spatial
`material`	scalar
	spatial, non spatial
`Resultflux`	scalar
	spatial

Operation

This operation calculates for each cell the accumulated amount of material that flows out of the cell into its neighbouring downstream cell. This accumulated amount is the amount of material in the cell itself plus the amount of material in upstream cells of the cell. For each cell, the following procedure is performed: using the local drain direction network on `ldd`, the catchment of a cell its outflow is determined which is made up the cell itself and all cells that drain to the cell (i.e. which are in upstream direction of the cell). The `material` values of all cells in the catchment are summed and send to the cell on `Resultflux`. This value is the amount of material which accumulates during transport in downstream direction to the outflow of the cell.

Notes

The values on `material` must be equal to or larger than zero. A cell with missing value on `material` is assigned a missing value on `Resultflux`. Additionally, all its downstream cells are assigned a missing value.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

Section 4.4.5 lddmask

Examples

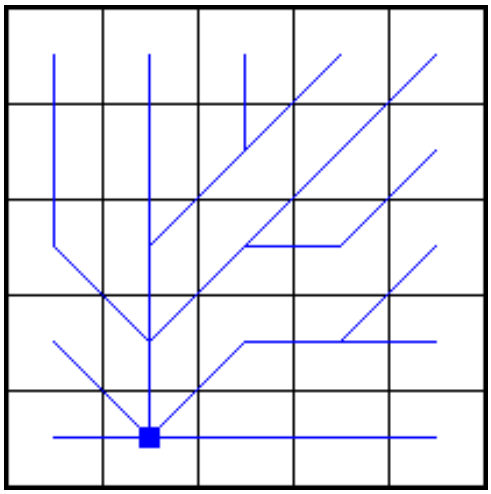
1.

```
pcrcalc  Result.map=accuflux(Ldd.map,Material.map)
```

Result.map

6	0.5	2	2	2
6.5	1	6	4	2
7	7.5	10	4	0
0.5	25	6	0	0
0.5	56	18	12	6

Ldd.map



Material.map

6	0.5	2	2	2
0.5	0.5	2	2	2
0.5	0.5	2	2	0
0.5	0.5	6	0	0
0.5	6	6	6	6

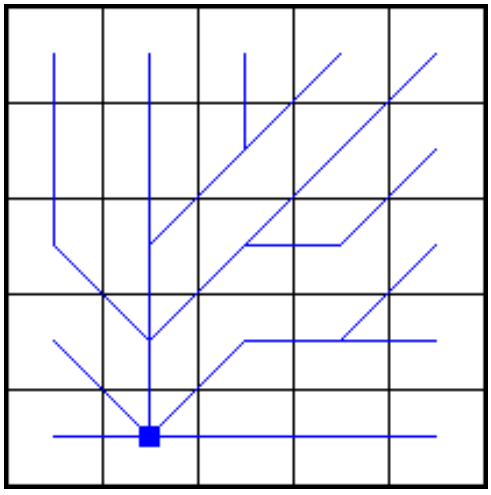
2.

```
pcrcalc  Result2.map=accuflux(Ldd.map,1)
```

Result2.map

1	1	1	1	1
2	2	3	2	1
3	6	5	2	1
1	15	4	3	1
1	25	3	2	1

Ldd.map



Prev accucapacityflux, accucapacitystate

Home Up

Next accufractionflux, accufractionstate

accufractionflux, accufractionstate

Name

accufractionflux, accufractionstate -- Fractional material transport downstream over local drain direction network

Synopsis

```
pcrcalc Resultflux = accufractionflux ( ldd, material, transportfraction )
pcrcalcResultstate = accufractionstate( ldd, material, transportfraction )
pcrcalcResultflux, Resultstate = accufractionflux, accufractionstate( ldd, material, transportfraction )
```

`ldd`	ldd
	spatial
`material`	scalar
	spatial, non spatial
`transportfraction`	scalar
	spatial, non spatial
`Resultflux`	scalar
	spatial
`Resultstate`	scalar
	spatial

Operation

These operations describe the accumulation of material in a drainage network with transport of a certain fraction. The remaining material is withdrawn from the stream. The operators enable the description of phenomena such as loss of a certain percentage of organic matter over a river stretch.

For each cell, **accufractionflux** assigns the amount of material which is transported out of the cell, **accufractionstate** assigns the amount which is stored in the cell. Both perform the same function of accumulation of material with a transport fraction, the only difference between the operators is the sort of result that is saved: **accufractionstate** yields storages of material in cells, **accufractionflux** yields fluxes of material out of cells.

For each cell, the amount of material input, for instance the amount of rain, is given by `material`. This is transported in downstream direction through the consecutively neighbouring downstream cells, following the local drain directions on `ldd`. Each time material moves through a cell a certain amount is stored in the cell. These storages are saved as `Resultstate`, if the **accufractionstate** operator is used. The remaining material is transported out of the cell, these amounts of outflow from each cell into its neighbouring downstream cell are the result of the **accufractionflux** operator, they are saved as `Resultflux`.

The function can be described by flow of material through a set of linked systems, where a cell represents a system. The flow starts at the cells/systems at the watershed boundaries (defined by `ldd`) and ends at a pit cell. The systems are linked by the local drain directions on `ldd`; these define the path of flow through the set of cells/systems. Each time a system is passed, the amount of flow changes.

For a cell/system somewhere on the map, the flow of material is described by a system. The inflow of the cell is the sum of the outflow amounts of its upstream neighbours. This inflow amount is added to the `material` value in the cell itself. This amount of material is potentially available for transport out of the cell. The amount actually transported is this amount multiplied by the `transportfraction` value of the cell. The remaining material is stored in the cell. Since `transportfraction` is a fraction it must contain values equal to or between 0 and 1 ([0,1]). (If `transportfraction` is 0 nothing will be transported, if it is 1 all material will be transported).

For each cell, the amount of material which is transported to its downstream neighbour (or out of the map if the cell is a pit cell) is saved as `Resultflux` (use the operator **accufractionflux**); the amount of material which is stored to the cell is saved as `Resultstate` (use **accufractionstate**)

Notes

The values on `material` must be equal to or larger than zero. The values on `transportfraction` must be equal to or between 0 and 1.

A cell with missing value on `material` and/or `transportfraction` is assigned a missing value on `Resultflux` or `Resultstate`. Additionally, all its downstream cells are assigned a missing value. The local drain direction network on `ldd` must be sound.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

Section 4.4.5 lddmask

Examples

1.
```
pcrcalc  Flux1.map,State1.map = accufractionflux, accufractionstate(Ldd.map,Material.map,0.5)
```

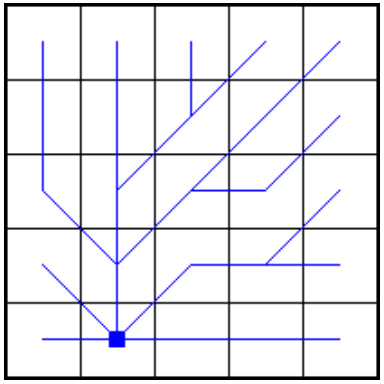
Flux1.map

3	0.25	1	1	1
1.75	0.375	2	1.5	1
1.12	1.44	2.5	1.5	0
0.25	2.78	3	0	0
0.25	8.77	5.25	4.5	3

State1.map

3	0.25	1	1	1
1.75	0.375	2	1.5	1
1.12	1.44	2.5	1.5	0
0.25	2.78	3	0	0
0.25	8.77	5.25	4.5	3

Ldd.map



Material.map

6	0.5	2	2	2
0.5	0.5	2	2	2
0.5	0.5	2	2	0
0.5	0.5	6	0	0
0.5	6	6	6	6

2.
```
pcrcalc  Flux2.map,State2.map =
 accufractionflux, accufractionstate(Ldd.map,Material.map,TransFra.map)
```

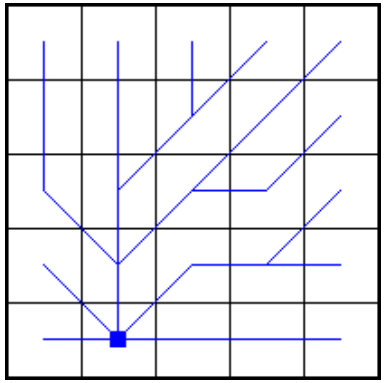
Flux2.map

9	10	1	1	1
17.1	20	1.2	1.1	1
24.4	31.2	1.22	1.1	1
9	66.8	1.16	1.6	5
45	178	5.55	5.49	4.9

State2.map

1	0	9	9	9
1.9	0	10.8	9.9	9
2.71	0	11	9.9	9
1	0	10.4	14.4	45
5	0	49.9	49.4	44.1

Ldd.map



Material.map

10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	50
50	50	50	50	49

TransFra.map

0.9	1	0.1	0.1	0.1
0.9	1	0.1	0.1	0.1
0.9	1	0.1	0.1	0.1
0.9	1	0.1	0.1	0.1
0.9	1	0.1	0.1	0.1

Prev accuflux
Home
Up
Next accuthresholdflux, accuthresholdstate

accuthresholdflux, accuthresholdstate

Name

accuthresholdflux, accuthresholdstate -- Input of material downstream over a local drain direction network when transport threshold is exceeded

Synopsis

```
pcrcalc Resultflux = accuthresholdflux ( ldd, material, transportthreshold )
pcrcalcResultstate = accuthresholdstate( ldd, material, transportthreshold )
pcrcalcResultflux, Resultstate = accuthresholdflux, accuthresholdstate( ldd, material, transportthreshold )
```

`ldd`	ldd
	spatial
`material`	scalar
	spatial, non spatial
`transportthreshold`	scalar
	spatial, non spatial
`Resultflux`	scalar
	spatial
`Resultstate`	scalar
	spatial

Operation

These operations describe accumulation of material in a drainage network with transport limited by a threshold: transport will only occur if a certain threshold of losses has been reached. Material less than the threshold is stored. This is the case for overland flow which will only develop once a certain loss has occurred, saturating the soil. The mechanism can also be used to describe phenomena such as losses from the streamflow due to infiltration of river water through the riverbed.

For each cell, **accuthresholdflux** assigns the amount of material which is transported out of the cell, **accuthresholdstate** assigns the amount which is stored in the cell. Both operators perform the same function of accumulation of material with a transport threshold, the only difference between the operators is the sort of result that is saved: **accuthresholdstate** yields storages of material in cells, **accuthresholdflux** yields fluxes of material out of cells.

For each cell, the amount of material input, for instance the amount of rain, is given by `material`. This is transported in downstream direction through the consecutively neighbouring downstream cells, following the local drain directions on `ldd`. Each time material moves through a cell an certain amount is stored in the cell. These storages are saved as `Resultstate`, if the **accuthresholdstate** operator is used. The remaining material is transported out of the cell, these amounts of outflow from each cell into its neighbouring downstream cell are the result of the **accuthresholdflux** operator, they are saved as `Resultflux`.

The function can be described by flow of material through a set of linked systems, where a cell represents a system. The flow starts at the cells/systems at the watershed boundaries (defined by `ldd`) and ends at a pit cell. The systems are linked by the local drain directions on `ldd`, these define the path of flow through the set of cells/systems. Each time a system is passed, the amount of flow changes.

For a cell/system somewhere on the map, the flow of material is described by a system. The inflow of the cell is the sum of the outflow amounts of its upstream neighbours. This inflow amount is added to the `material` value in the cell itself. This amount of material is potentially available for transport out of the cell. If it is less than or equal to the `transportthreshold` value of the cell all material is stored. If it is more than the `transportthreshold` the amount transported is the amount potentially available for transport minus the `transportthreshold` value. The

remaining material is stored.

For each cell, the amount of material which is transported to its downstream neighbour (or out of the map if the cell is a pit cell) is saved as `Resultflux` (use the operator **accuthresholdflux**); the amount of material which is stored to the cell is saved as `Resultstate` (use **accuthresholdstate**)

Notes

The values on `material` and `transportthreshold` must be equal to or larger than zero.

A cell with missing value on `material` and/or `transportthreshold` is assigned a missing value on `Resultflux` or `Resultstate`. Additionally, all its downstream cells are assigned a missing value.

The local drain direction network on `ldd` must be sound.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

Section 4.4.5 lddmask

Examples

1.

```
pcrcalc  Flux1.map, State1.map = accuthresholdflux, accuthresholdstate(Ldd.map, Material.map, 1.5)
```

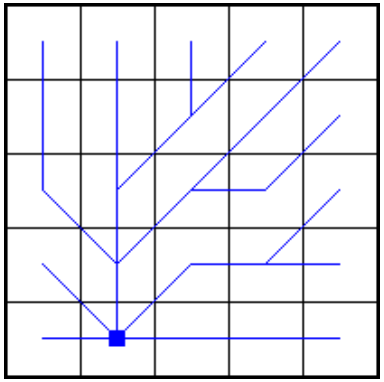
`Flux1.map`

4.5	0	0.5	0.5	0.5
3.5	0	1.5	1	0.5
2.5	0.5	2.5	1	0
0	4.5	4.5	0	0
0	27	13.5	9	4.5

`State1.map`

1.5	0.5	1.5	1.5	1.5
1.5	0.5	1.5	1.5	1.5
1.5	1.5	1.5	1.5	0
0.5	1.5	1.5	0	0
0.5	1.5	1.5	1.5	1.5

`Ldd.map`



`Material.map`

6	0.5	2	2	2
0.5	0.5	2	2	2
0.5	0.5	2	2	0
0.5	0.5	6	0	0
0.5	6	6	6	6

2.

```
pcrcalc  Flux2.map, State2.map = accuthresholdflux, accuthresholdstate(Ldd.map, Material.map, TransTH.map)
```

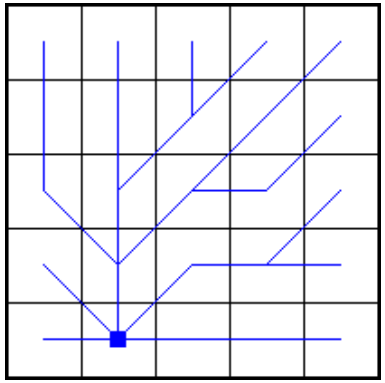
Flux2.map

10	0	1	1	1
20	0	3	2	1
30	0	0	2	1
10	0	3	2	0
50	93	20	10	0

State2.map

0	10	9	9	9
0	10	9	9	9
0	13	14	9	9
0	40	9	9	50
0	40	40	40	49

Ldd.map



Material.map

10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	50
50	50	50	50	49

TransTH.map

0	40	9	9	9
0	40	9	9	9
0	40	30	9	9
0	40	9	9	50
0	40	40	40	50

Prev
accufractionflux, accufractionstate

Home
Up

Next
accutriggerflux, accutriggerstate

accutriggerflux, accutriggerstate

Name

accutriggerflux, accutriggerstate -- Input of material downstream over a local drain direction network when transport trigger is exceeded

Synopsis

```
pcrcalc Resultflux = accutriggerflux ( ldd, material, transporttrigger )
pcrcalcResultstate = accutriggerstate( ldd, material, transporttrigger )
pcrcalcResultflux, Resultstate = accutriggerflux, accutriggerstate( ldd, material, transporttrigger )
```

`ldd`	ldd
	spatial
`material`	scalar
	spatial, non spatial
`transporttrigger`	scalar
	spatial, non spatial
`Resultflux`	scalar
	spatial
`Resultstate`	scalar
	spatial

Operation

These operations describe accumulation of material in a drainage network with transport limited by a trigger: transport occurs once a trigger value has been exceeded. When the trigger value is not exceeded, no transport takes place and all material is stored. This may be the case with landslides, where the soil has to be saturated first before all water (and soil) will be transported downhill, or avalanches where the total amount of snow will come down once triggered.

Both operators perform the same function of accumulation of material with a transport trigger, the only difference between the operators is the sort of result that is saved: **accutriggerstate** yields storages of material in cells, **accutriggerflux** yields fluxes of material out of cells.

For each cell, the amount of material input, for instance the amount of rain, is given by `material`. This is transported in downstream direction through the consecutively neighbouring downstream cells, following the local drain directions on `ldd`. Each time material moves through a cell an certain amount is stored in the cell. These storages are saved as `Resultstate`, if the **accutriggerstate** operator is used. The remaining material is transported out of the cell, these amounts of outflow from each cell into its neighbouring downstream cell are the result of the **accutriggerflux** operator, they are saved as `Resultflux`.

The function can be described by flow of material through a set of linked systems, where a cell represents a system. The flow starts at the cells/systems at the watershed boundaries (defined by `ldd`) and ends at a pit cell. The systems are linked by the local drain directions on `ldd`, these define the path of flow through the set of cells/systems. Each time a system is passed, the amount of flow changes.

For a cell/system somewhere on the map, the flow of material is described by a system. The inflow of the cell is the sum of the outflow amounts of its upstream neighbours. This inflow amount is added to the `material` value in the cell itself. This amount of material is potentially available for transport out of the cell. If it is less than or equal to the `transporttrigger` value of the cell all material is stored. If it is more than the `transporttrigger` all material is transported and nothing is stored.

For each cell, the amount of material which is transported to its downstream neighbour (or out of the map if the cell is a pit cell) is saved as `Resultflux` (use the operator **accutriggerflux**); the

amount of material which is stored to the cell is saved as `Resultstate` (use **accutriggerstate**)

Notes

The values on `material` and `transporttrigger` must be equal to or larger than zero.

A cell with missing value on `material` and/or `transporttrigger` is assigned a missing value on `Resultflux` or `Resultstate`. Additionally, all its downstream cells are assigned a missing value.

The local drain direction network on `ldd` must be sound.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

Section 4.4.5 lddmask

Examples

1.

```
pcrcalc  Flux1.map, State1.map = accutriggerflux, accutriggerstate(Ldd.map, Material.map, 1.5)
```

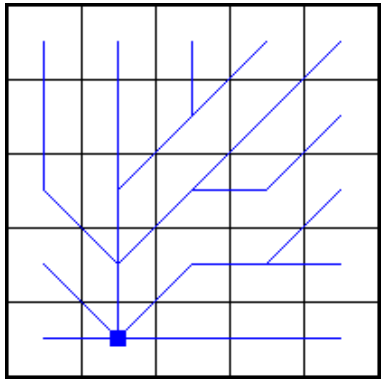
Flux1.map

6	0	2	2	2
6.5	0	6	4	2
7	6.5	10	4	0
0	24	6	0	0
0	54	18	12	6

State1.map

0	0.5	0	0	0
0	0.5	0	0	0
0	0	0	0	0
0.5	0	0	0	0
0.5	0	0	0	0

Ldd.map



Material.map

6	0.5	2	2	2
0.5	0.5	2	2	2
0.5	0.5	2	2	0
0.5	0.5	6	0	0
0.5	6	6	6	6

2.

```
pcrcalc  Flux2.map, State2.map = accutriggerflux, accutriggerstate(Ldd.map, Material.map, TransTH.map)
```

accutriggerflux, accutriggerstate

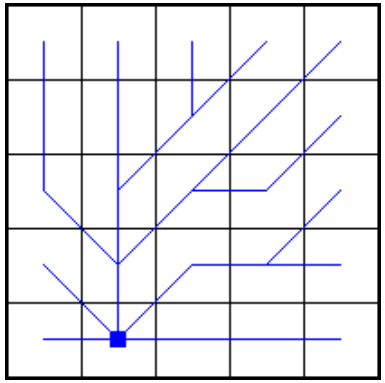
Flux2.map

10	0	10	10	10
20	0	30	20	10
30	40	50	20	10
10	130	80	70	50
50	420	100	50	0

State2.map

0	10	0	0	0
0	10	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	49

Ldd.map



Material.map

10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	50
50	50	50	50	49

TransTH.map

0	40	9	9	9
0	40	9	9	9
0	40	30	9	9
0	40	9	9	50
0	40	40	40	50

Prev
accuthresholdflux, accuthresholdstate

Home
Up

Next
acos

acos

Name

acos -- Inverse cosine

Synopsis

pcrcalc [option] Result = **acos** (expression)

expression	scalar
	spatial, non spatial
Result	direction
	dimension of expression

Options

affects Result only if expression is a number:

--degrees

direction is given in degrees (default)

--radians

direction is given in radians

Operation

This operator calculates for each cell the inverse cosine of the cell value on expression and assigns it to Result.

Notes

The values on expression must be equal to or between -1 and 1. Cells with a value outside this range will be assigned a missing value on Result. A cell with missing value on expression is assigned a missing value on Result.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.

```
pcrcalc  --degrees Result.map = acos(Expr.map)
```

Result.map

60	MV	120
MV	MV	90
104	84.3	53.1

Expr.map

0.5	MV	-0.5
-1.1	3	0
-0.25	0.1	0.6

and

Name

and -- Boolean-AND operation

Synopsis

pcrcalc Result = expression1 **and** expression2

expression1	boolean
	spatial, non spatial
expression2	boolean
	spatial, non spatial
Result	boolean
	spatial; non spatial if expression1 and expression2 are non spatial

Operation

The cell values on expression1 and expression2 are interpreted as Boolean values; where 1 is TRUE and 0 is FALSE. For each cell the Boolean AND evaluation is performed: if both expression1 and expression2 have a cell value 1 (TRUE) Result has a cell value 1 (TRUE) on the corresponding cell; if expression1 or expression2 or both have a cell value 0 (FALSE) Result has cell value 0 (FALSE).

Table 1. Cross table of the AND operator.

AND		**expression1**	
		False	True
expression2	False	False	False
	True	False	True

Notes

A cell with missing value on `expression1` or `expression2` or on both expressions results in a missing value on `Result` at the corresponding cell.

Group

This operation belongs to the group of Boolean operators

Examples

1.

```
pcrcalc  Result.map = Expr1.map and Expr2.map
```

`Result.map`

0	1	0
0	MV	MV
1	1	0

`Expr1.map`

1	1	0
0	MV	0
1	1	0

`Expr2.map`

0	1	0
1	1	MV
1	1	1

areaarea

Name

areaarea -- The area of the area to which a cell belongs

Synopsis

pcrcalc [option] Result = **areaarea** (areaclass)

areaclass	boolean, nominal, ordinal
	spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell

--unittrue

area is computed in true area represented by cells (default)

--unitcell

area is computed in number of cells

Operation

The class to which a cell belongs is identified by areaclass: all cells with corresponding values on areaclass are grouped and together they form one class. For each separate class the total area that is represented by the cells belonging to that class is calculated (cell value times the total number of classes). This value is assigned to all cells belonging to that class. This is done for all classes and saved as Result.

Notes

A cell with missing value on `areaclass` is assigned a missing value on `Result` at the corresponding cell.

Group

This operation belongs to the group of Area operators

See Also

Section 4.5

Examples

1.

```
pcrcalc  Result.map=areaarea(Class.map)
```

`Result.map`

24	32	24	24	MV
32	32	24	24	24
32	32	24	24	24
32	32	24	24	24
32	8	8	8	8

`Class.map`

2	6	2	2	MV
6	6	2	2	2
6	6	0	0	0
6	6	0	0	0
6	3	3	4	4

areaaverage

Name

areaaverage -- Average cell value of within an area

Synopsis

pcrcalc Result = **areaaverage** (expression, areaclass)

expression	scalar
	spatial
areaclass	boolean, nominal, ordinal
	spatial
Result	scalar
	spatial

Operation

areaclass Identifies the class to which a cell belongs: cells with corresponding values on areaclass together form a separate class. For each separate class the expression values of the cells belonging to that class are averaged. This average value is assigned to all cells belonging to that class. This is done for all classes and saved as Result.

Notes

A cell with missing value on areaclass will result in a missing value on Result at the corresponding cell.

Group

This operation belongs to the group of Area operators

See Also

Section 4.5

Examples

1.
```
pcrcalc  Result.map=areaaverage( Expr.map, Class.map)
```

Result.map

1.2	0	1.2	1.2	MV
0	0	1.2	1.2	1.2
0	0	0.133	0.133	0.133
0	0	0.133	0.133	0.133
0	2.5	2.5	10	10

Expr.map

0	0	0	MV	3
0	0	0	3	3
0	0	0	0.2	0.2
0	0	0	0.2	0.2
0	-2	7	7	13

Class.map

2	6	2	2	MV
6	6	2	2	2
6	6	0	0	0
6	6	0	0	0
6	3	3	4	4

Prev areaarea | Home | Up | Next areadiversity

areadiversity

Name

areadiversity -- Number of unique cell values within an area

Synopsis

pcrcalc Result = **areadiversity** (expression, areaclass)

expression	ordinal
	spatial
areaclass	boolean, nominal, ordinal
	spatial
Result	scalar
	spatial

Operation

areaclass Identifies the class to which a cell belongs: cells with corresponding values on areaclass are member of a separate class. For each separate class the number of unique cell values on expression is counted. This number is assigned to all cells belonging to that class. This is done for all classes and saved as Result.

Notes

A cell with missing value on areaclass will result in a missing value on Result at the corresponding cell.

Group

This operation belongs to the group of Area operators

See Also

Section 4.5

Examples

1.
```
pcrcalc  Result.map=areadiversity( Expr.map, Class.map)
```

Result.map

1	5	1	1	MV
5	5	1	1	1
5	5	6	6	6
5	5	6	6	6
5	2	2	1	1

Expr.map

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
7	6	0	4	18
2	6	2	8	-1
5	2	3	3	MV

Class.map

2	6	2	2	MV
6	6	2	2	2
6	6	0	0	0
6	6	0	0	0
6	3	3	4	4

Prev areaaverage | Home Up | Next areamajority

areamajority

Name

areamajority -- Most often occurring cell value within an area

Synopsis

pcrcalc Result = **areamajority** (expression, areaclass)

expression	boolean, nominal, ordinal spatial
areaclass	boolean, nominal, ordinal spatial
Result	type of expression spatial

Operation

areaclass Identifies the class to which a cell belongs: cells with corresponding values on areaclass are member of a separate class. For each separate class the most often occurring cell value on expression is determined. This value is assigned to all cells belonging to that class. This is done for all classes and saved as Result. If two values both occur the same number of times, the largest value of these values is assigned.

Notes

A cell on areaclass with missing value will result in a missing value on Result at the corresponding cell.

Group

This operation belongs to the group of Area operators

See Also

Section 4.5

Examples

1.
```
pcrcalc  Result.map=areamajority(Expr.map, Class.map)
```

Result.map

1	-6	1	1	MV
-6	-6	1	1	1
-6	-6	0	0	0
-6	-6	0	0	0
-6	4	4	1	1

Expr.map

MV	1	1	1	1
-6	-6	18	1	0
-6	-6	-6	0	0
-6	-6	-6	0	4
0	4	0	1	-6

Class.map

2	6	2	2	MV
6	6	2	2	2
6	6	0	0	0
6	6	0	0	0
6	3	3	4	4

Prev areadiversity
Home Up
Next areamaximum

areamaximum

Name

areamaximum -- Maximum cell value within an area

Synopsis

pcrcalc Result = **areamaximum** (expression, areaclass)

expression	ordinal, scalar spatial
areaclass	boolean, nominal, ordinal spatial
Result	type of expression spatial

Operation

areaclass Identifies the class to which a cell belongs: cells with corresponding values on areaclass are member of a separate class. For each separate class the maximum expression value of the cells belonging to that class is determined. This value is assigned to all cells belonging to that class. This is done for all classes and saved as Result.

Notes

A cell with a missing value areaclass is assigned a missing value on Result at the corresponding cell.

Group

This operation belongs to the group of Area operators

See Also

Section 4.5

Examples

1.
```
pcrcalc  Result.map=areamaximum(Expr.map, Class.map)
```

Result.map

-6	8	-6	-6	MV
8	8	-6	-6	-6
8	8	8	8	8
8	8	8	8	8
8	1	1	2.5	2.5

Expr.map

-9	0	-6	-6	-6
1	1	-6	-6	MV
1	1	-1	7	2
1	1	3	5	8
8	1	1	2.5	1.4

Class.map

2	6	2	2	MV
6	6	2	2	2
6	6	0	0	0
6	6	0	0	0
6	3	3	4	4

Prev areamajority | Home Up | Next areaminimum

areaminimum

Name

areaminimum -- Minimum cell value within an area

Synopsis

pcrcalc `Result = areaminimum ( expression, areaclass )`

`expression`	ordinal, scalar spatial
`areaclass`	boolean, nominal, ordinal spatial
`Result`	type of `expression` spatial

Operation

`areaclass` Identifies the class to which a cell belongs: cells with corresponding values on `areaclass` are member of a separate class. For each separate class the minimum `expression` value of the cells belonging to that class is determined. This value is assigned to all cells belonging to that class. This is done for all classes and saved as `Result`.

Notes

A cell with missing value on `areaclass` is assigned a missing value on `Result` at the corresponding cell.

Group

This operation belongs to the group of Area operators

See Also

Section 4.5

Examples

1.
```
pcrcalc  Result.map=areaminimum(Expr.map, Class.map)
```

Result.map

-9	0	-9	-9	MV
0	0	-9	-9	-9
0	0	-1	-1	-1
0	0	-1	-1	-1
0	1	1	1.4	1.4

Expr.map

-9	0	-6	-6	-6
1	1	-6	-6	MV
1	1	-1	7	2
1	1	3	5	8
8	1	1	2.5	1.4

Class.map

2	6	2	2	MV
6	6	2	2	2
6	6	0	0	0
6	6	0	0	0
6	3	3	4	4

Prev areamaximum | Home Up | Next areanormal

areanormal

Name

areanormal -- Value assigned to an area taken from a normal distribution

Synopsis

pcrcalc Result = **areanormal** (areaclass)

areaclass	boolean,nominal,ordinal
	spatial
Result	scalar
	spatial

Operation

The area to which a cell belongs is identified by areaclass: cells with corresponding values on areaclass are member of a separate area. The Result is generated with a random number generator: for each area on areaclass, a random number is taken from a normal distribution with mean 0 and standard deviation 1. This value is assigned to all cells belonging to that area.

Notes

A cell with a missing value on areaclass is assigned a missing value on Result.

Group

This operation belongs to the group of Random number generators; Areas

See Also

Section 6.1.11 Section 6.4.2

Examples

1.

```
pcrcalc  Result.map = areanormal(Class.map)
```

Result.map

0.647	0.494	0.647	0.647	MV
0.494	0.494	0.647	0.647	0.647
0.494	0.494	0.764	0.764	0.764
0.494	0.494	0.764	0.764	0.764
0.494	-1.15	-1.15	-0.517	0.517

Class.map

2	6	2	2	MV
6	6	2	2	2
6	6	0	0	0
6	6	0	0	0
6	3	3	4	4

Prev areaminimum | Home | Up | Next areatotal

areatotal

Name

areatotal -- Sum of cell values within an area

Synopsis

pcrcalc Result = **areatotal**(expression, areaclass)

`expression`	scalar
	spatial
`areaclass`	boolean, nominal, ordinal
	spatial
`Result`	scalar
	spatial

Operation

`areaclass` Identifies the class to which a cell belongs: cells with corresponding values on `areaclass` are member of a separate class. For each separate class the `expression` values of the cells belonging to that class are summed. This sum is assigned to all cells belonging to that class. This is done for all classes and saved as `Result`.

Notes

A cell with missing value on `areaclass` is assigned a missing value on `Result` at the corresponding cell.

Group

This operation belongs to the group of Area operators

See Also

Section 4.5

Examples

1.
```
pcrcalc  Result.map=areatotal( Expr.map, Class.map)
```

Result.map

-33	14	-33	-33	MV
14	14	-33	-33	-33
14	14	24	24	24
14	14	24	24	24
14	2	2	3.9	3.9

Expr.map

-9	0	-6	-6	-6
1	1	-6	-6	MV
1	1	-1	7	2
1	1	3	5	8
8	1	1	2.5	1.4

Class.map

2	6	2	2	MV
6	6	2	2	2
6	6	0	0	0
6	6	0	0	0
6	3	3	4	4

Prev areanormal
Home Up
Next areauniform

areauniform

Name

areauniform -- Value assignseds to area taken from an uniform distribution

Synopsis

pcrcalc Result = **areauniform**(areaclass)

areaclass	boolean,nominal,ordinal
	spatial
Result	scalar
	spatial

Operation

The area to which a cell belongs is identified by areaclass: cells with corresponding values on areaclass are member of a separate area. The Result is generated with a random number generator: for each area on areaclass, a random number between 0 and 1 is taken from a uniform distribution. This value is assigned to all cells belonging to that area.

Notes

A cell with a missing value on areaclass is assigned a missing value on Result.

Group

This operation belongs to the group of Random number generators; Areas

See Also

Section 6.1.11 Section 6.4.2

Examples

1.

```
pcrcalc  Result.map = areauniform(Class.map)
```

Result.map

0.401	0.936	0.401	0.401	MV
0.936	0.936	0.401	0.401	0.401
0.936	0.936	0.348	0.348	0.348
0.936	0.936	0.348	0.348	0.348
0.936	0.905	0.905	0.945	0.945

Class.map

2	6	2	2	MV
6	6	2	2	2
6	6	0	0	0
6	6	0	0	0
6	3	3	4	4

Prev areatotal | Home | Up | Next asin

asin

Name

asin -- Inverse sine

Synopsis

pcrcalc [option] Result = **asin**(expression)

expression	scalar
	spatial, non spatial
Result	directional
	dimension of expression

Options

if expression is a number: --degrees or --radians

--degrees

direction is given in degrees (default)

--radians

direction is given in radians

Operation

For each cell, calculates the inverse cosine of the cell value on expression and assigns it to Result.

Notes

The values on expression must be equal to or between -1 and 1. Cells with a value outside this range will be assigned a missing value on Result. A cell with missing value on expression is assigned a missing value on Result.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.

```
pcrcalc  Result.map = asin(Expr.map)
```

Result.map

53.1	330	MV
30	270	13.3
MV	MV	0

Expr.map

0.8	-0.5	MV
0.5	-1	0.23
-7	1.2	0

aspect

Name

aspect -- Aspects of a map using a digital elevation model

Synopsis

```
pcrcalc Result = aspect( dem )
```

`dem`	scalar
	spatial
`Result`	directional
	spatial

Operation

For each cell, calculates the direction of maximum rate of change in elevation (aspect) on basis of the elevation of its 8 neighbours in a 3 x 3 cells window. The third-order finite difference method is used, proposed by [horn81], also used by [skidmore89]. `Result` is of a directional data type, with aspect values assigned clockwise; aspect to the top of the map is taken as zero aspect. When using **display** the aspect will be expressed in degrees or radians (depending on the global setting --degrees (default) or --radians).

Notes

If a cell has a missing value on `dem`, a missing value is assigned to `Result`, in any case.

For each cell, the aspect is calculated using its 8 neighbours in a 3 x 3 cells window. Elevation in all these cells must be known, else the finite difference method can not be performed. It may occur that one of these values is unknown: this is the case if a surrounding cell is a missing value or if the centre cell is at the edge of the map resulting in the absence of some surrounding cells. If this occurs the **aspect** operator uses a built in neighbourhood interpolator to fill these missing values or absent cells in; this will make calculation of the slope for the centre cell still possible. For each missing value cell or absent cell, the elevation is determined by taking the average elevation of non missing value cells in a 3 x 3 cell window, with the missing value cell or absent cell in the centre of the window.

Group

This operation belongs to the group of Derivatives of elevation maps

See Also

nodirection

Examples

1.

```
pcrcalc  Result.map = aspect(Dem.map)
```

Result.map

225	280	293	MV	331
45	284	290	MV	MV
18.7	358	307	288	308
4.93	13.1	358	320	324
360	28.5	5.73	324	334

Dem.map

70	70	80	MV	120
70	70	90	MV	MV
70	70	100	140	280
180	160	110	160	320
510	440	300	400	480

atan

Name

atan -- Inverse tangent

Synopsis

pcrcalc [option] Result = **atan**(expression)

expression	scalar
	spatial, non spatial
Result	directional
	dimension of expression

Options

if expression is a number: --degrees or --radians

--degrees

direction is given in degrees (default)

--radians

direction is given in radians

Operation

For each cell, calculates the inverse tangent of the cell value on expression and assigns it to Result.

Notes

A cell with missing value on expression is assigned a missing value on Result.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.

```
pcrcalc  Result.map = atan(Expr.map)
```

Result.map

0	MV	88.8
315	272	63.4
5.71	338	71.6

Expr.map

0	MV	47
-1	-30	2
0.1	-0.4	3

Prev aspect | Home | Up | Next boolean

boolean

Name

boolean -- Conversion data type to boolean data type

Synopsis

pcrcalc `Result = ` **boolean**`( expression )`

`expression`	nominal, ordinal, scalar, directional, ldd spatial, non spatial
`Result`	boolean dimension of `expression`

Operation

If `expression` is a PCRaster map or a calculation resulting in a PCRaster map of one of the data types nominal, ordinal, scalar, directional or ldd, converts to a Boolean data type with Boolean cell values, where 1 is TRUE and 0 is FALSE. Each `expression` cell value not equal to 0 is assigned a 1 (TRUE) on `Result`; each `expression` cell value equal to 0 is assigned a 0 (FALSE) on `Result`. Or it generates a map of boolean data type with one constant value.

If `expression` has no PCRaster data type, **boolean** generates a boolean `Result`. This is the case if `expression` is a number. This number must be in the domain of the boolean map type, i.e. 0 or 1. `Result` will be a map with the same location attributes as the global clone map ; all cells will have the value of `expression`.

Notes

A cell with missing value on `expression` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Conversion and assignment

See Also

Section 2.3.3 Section 2.3.3.2

Examples

1.

```
pcrcalc  Result.map = boolean(Expr.map)
```

Result.map

1	0	1
MV	1	1
1	1	0

Expr.map

-10	0	0.01
MV	350	21
-1	400	0

catchment

Name

catchment -- Catchment(s) of one or more specified cells

Synopsis

```
pcrcalc Result = catchment( ldd, points )
```

`ldd`	ldd spatial
`points`	boolean, nominal, ordinal spatial
`Result`	type of `points` spatial

Operation

The local drain direction for each cell is defined by `ldd`. For each non zero value on `points` its catchment is determined and all cells in its catchment are assigned this non zero value. This procedure is performed for all cells with a non zero value on `points`, but there is one important exception: subcatchments are not identified: if the catchment of a non zero cell on `points` is a part of another (larger) catchment of a non zero cell on `points`, the cells in this smaller subcatchment are assigned the value of the larger enclosing catchment.

The operation is performed as follows: for each cell its downstream path is determined which consists of the consecutively neighbouring downstream cells on `ldd`. On `Result` each cell is assigned the non zero `points` cell value which is on its path and which is most far downstream. If all cells on the downstream path of a cell have a value 0 on `points` a 0 is assigned to the cell on `Result`.

Notes

A cell with missing value on `ldd` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

subcatchment lddmask

Examples

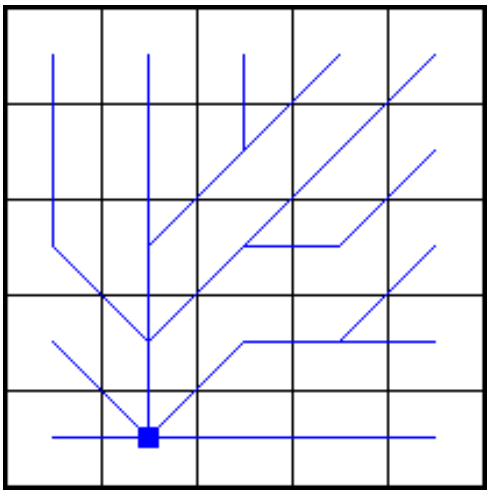
1.

```
pcrcalc  Result.map = catchment(Ldd.map,Points.map)
```

Result.map

3	3	3	3	3
3	3	3	3	3
3	3	3	3	2
0	3	2	2	2
0	0	0	0	1

Ldd.map



Points.map

0	0	0	0	0
0	0	0	MV	0
0	4	0	0	0
0	3	2	0	0
0	0	0	0	1

Prev boolean | Home Up | Next catchmenttotal

catchmenttotal

Name

catchmenttotal -- Total catchment for the entire upstream area

Synopsis

pcrcalc Result = **catchmenttotal**(amount,ldd)

amount	scalar
	spatial, non spatial
ldd	ldd
	spatial
Result	scalar
	spatial

Operation

This operation is identical to accuflux, except that accuflux does not accept negative values. Catchmenttotal calculates for each cell the accumulated amount of material that flows out of the cell into its neighbouring downstream cell. This accumulated amount is the amount of material in the cell itself plus the amount of material in upstream cells of the cell. For each cell, the following procedure is performed: using the local drain direction network on `ldd`, the catchment of a cell its outflow is determined which is made up the cell itself and all cells that drain to the cell (i.e. which are in upstream direction of the cell). The `material` values of all cells in the catchment are summed and send to the cell on `Resultflux`. This value is the amount of material which accumulates during transport in downstream direction to the outflow of the cell.

cellarea

Name

cellarea -- Area of one cell

Synopsis

pcrcalc [option] Result = **cellarea** ()

`Result` scalar

non spatial

Options

--unittrue or --unitcell:

--unittrue

area is computed in true area; `Result` is true area of a cell (default)

--unitcell

area is computed in number of cells; `Result` is 1

Operation

Calculates the area represented by one cell and assigns this cell area to `Result` (non spatial).

Notes

Group

This operation belongs to the group of Map operators

Examples

1.

```
pcrcalc  --unittrue --clone Expr.map Result1.map = cellarea()
```

Result1.map

16	16	16
16	16	16
16	16	16

Expr.map

1	2	-6
5	4	3
2	8	MV

2.

```
pcrcalc  --unitcell --clone Expr.map Result2.map = cellarea()
```

Result2.map

1	1	1
1	1	1
1	1	1

Expr.map

1	2	-6
5	4	3
2	8	MV

Prev catchmenttotal | Home Up | Next celllength

celllength

Name

celllength -- Horizontal and vertical length of a cell

Synopsis

pcrcalc [option] Result = **celllength** ()

`Result` scalar

non spatial

Options

--unittrue or --unitcell

--unittrue

length of cells is computed in true length of cells; `Result` is true length of a cell (default)

--unitcell

length of cells is computed in unit cell length; `Result` is 1

Operation

Calculates the length (horizontal or vertical) of one cell and assigns this cell length to `Result` (non spatial).

Notes

Group

This operation belongs to the group of Map operators

Examples

1.
```
pcrcalc  --unittrue --clone Expr.map Result1.map = celllength()
```

Result1.map

4	4	4
4	4	4
4	4	4

Expr.map

1	2	-6
5	4	3
2	8	MV

2.
```
pcrcalc  --unitcell --clone Expr.map Result2.map = celllength()
```

Result2.map

1	1	1
1	1	1
1	1	1

Expr.map

1	2	-6
5	4	3
2	8	MV

Prev cellarea | Home | Up | Next clump

clump

Name

clump -- Contiguous groups of cells with the same value ('clumps')

Synopsis

pcrcalc [option] Result = **clump**(expression)

expression	boolean, nominal, ordinal
	spatial
Result	nominal
	spatial

Options

--diagonal or --nondiagonal

--diagonal

cells of the same value are grouped if they are within the immediate 8-cell neighbourhood of each other. This includes if they are to the right or left, above or below, or are diagonal to each other (eight nearest neighbours, default).

--nondiagonal

cells of the same value are grouped only if the cells are directly to the right or left, or above or below each other (four nearest neighbours).

Operation

Cells that have the same value on expression and are neighbours are grouped. Every group of cells satisfying these conditions is assigned a unique value on Result. Cells without neighbours with the same value on expression are also assigned a unique value on Result. The kind of connectivity needed for cells to be neighbours is specified by the option.

Notes

Cells with a missing value on `expression` are assigned a missing value on `Result`.

Group

This operation belongs to the group of Area operators

Examples

1.

```
pcrcalc   Result1.map=clump( Expr.map)
```

`Result1.map`

1	1	2	2	3
2	2	1	1	2
4	4	2	2	MV
4	4	5	5	5
4	5	5	5	5

`Expr.map`

4	4	5	5	3
5	5	4	4	5
8	8	5	5	MV
8	8	-2	-2	-2
8	-2	-2	-2	-2

2.

```
pcrcalc  --nondiagonal Result2.map=clump( Expr.map)
```

`Result2.map`

1	1	2	2	3
4	4	5	5	6
7	7	8	8	MV
7	7	9	9	9
7	9	9	9	9

`Expr.map`

4	4	5	5	3
5	5	4	4	5
8	8	5	5	MV
8	8	-2	-2	-2
8	-2	-2	-2	-2

---

cos

Name

cos -- Cosine

Synopsis

pcrcalc [option] Result = **cos**(expression)

expression	directional, scalar
	spatial, non spatial
Result	scalar
	dimension of expression

Options

if expression is a number: --degrees or --radians

--degrees

direction is measured in degrees (default)

--radians

direction is measured in radians

Operation

For each cell, calculates the cosine of the cell value on expression and assigns it to Result.

Notes

A cell with a missing value on expression is assigned a missing value on Result.

If expression is of directional data type, a cell on expression without a direction (cell value -1) is assigned a missing value.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.

```
pcrcalc  Result.map = cos(Expr.map)
```

Result.map

0.5	0.5	MV
0.977	0.998	0.951
-1	1	1

Expr.map

60	300	MV
12.3	4	18
180	0	0

Prev clump | Home | Up | Next cover

cover

Name

cover -- Missing values substituted for values from one or more expression(s)

Synopsis

```
pcrcalc Result = cover( expression 1, expression 2,... expression n)
```

expression1-n	boolean, nominal, ordinal, scalar, directional; must all have the same data type spatial, non spatial
Result	type of expression1-n spatial, non spatial

Operation

This operator is used to cover missing values on an expression with values taken from one or more different expression(s). A (theoretically infinite) number of n expressions can be specified between the brackets in the command line, where expression1 is typed first and expressionn last. For each cell, the value on one of the expression1, expression2,...expressionn is selected and assigned to Result. Per cell, the value is assigned of the first expression between the brackets in the command line with a non missing value (i.e. the value on the expression with the smallest n, omitting expressions with a missing value on the cell under consideration). If all expressions have a missing value, a missing value is assigned to Result.

Notes

Using **cover** for covering expression1, expression2,...expressionn of data type ldd is quite risky: possibly it will result in a ldd which is unsound. If you do want to cover expressions of data type ldd use the operator **lddrepair** afterwards. This operator will modify the ldd in such a way that it will be sound, see the operator **lddrepair**.

Group

This operation belongs to the group of Missing value creation

See Also

lddrepair

Examples

1.

```
pcrcalc  Result1.map = cover(Expr1.map,sqrt(9))
```

Result1.map

3	3	3
3	3	4
3	5	-1

Expr1.map

MV	MV	MV
MV	MV	4
MV	5	-1

2.

```
pcrcalc  Result2.map = cover(Expr1.map,Expr2.map,Expr3.map)
```

Result2.map

0	4	4
4	4	4
MV	5	-1

Expr1.map

MV	MV	MV
MV	MV	4
MV	5	-1

Expr2.map

0	MV	MV
MV	MV	18
MV	2.6	MV

Expr3.map

4	4	4
4	4	4
MV	4	4

defined

Name

defined -- Boolean TRUE for non missing values and FALSE for missing values

Synopsis

pcrcalc Result = **defined**(expression)

expression	boolean, nominal, ordinal, scalar, directional, ldd
	spatial, non spatial
Result	boolean
	dimension of expression

Operation

For each cell on Result returns a Boolean value where 1 is TRUE and 0 is FALSE: if the cell value on expression is not a missing value a 1 (TRUE) is assigned to the corresponding cell on Result, if the cell value on expression is a missing value a 0 (FALSE) is assigned to the corresponding cell on Result.

Notes

Group

This operation belongs to the group of Missing value creation

Examples

1.

```
pcrcalc  Result.map = defined(Expr.map)
```

defined

Result.map

1	0	1
1	1	1
1	0	1

Expr.map

4	MV	-4
-4	2	3.8
5	MV	0

Prev cover | Home | Up | Next directional

directional

Name

directional -- Data conversion to the directional data type

Synopsis

pcrcalc [option] Result = **directional**(expression)

expression	boolean, nominal, ordinal, scalar, ldd
	spatial, non spatial
Result	directional
	dimension of expression

Options

--degrees or --radians

--degrees

values on expression are interpreted as degrees (default)

--radians

values on expression are interpreted as radians

Operation

If expression is a PCRaster map or an calculation resulting in a PCRaster map, it is converted: if expression is of one of the data types boolean, nominal, ordinal, scalar the cell values on expression are converted to the circular scale of Result, on a cell-by-cell basis. If expression is of the data type ldd, the codes on expression representing local drain directions are converted to real directions of drainage and saved as Result. The directions are converted to the circular scale of Result clockwise, assigning 0 degrees (or radians if the option --radians is set) to cells with a local drainage towards the top of the map (ldd code 8). A flat cell (cell value 5 on expression) is assigned a -1 on Result. The command can also generate a map of directional data type with one constant value.

If expression has no PCRaster data type, a Result with data type directional is generated. This is the case if expression is a number or a calculation with numbers. The value of expression must

be in the domain of the directional data type, i.e. if the option --degrees is set: equal to 0 or between 0 and 360; if the option radians is set equal to 0 or between 0 and 2pi. `Result` will be a map with the same location attributes as the global clone map; all cells will have the value of `expression`.

Notes

A cell with missing value on `expression` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Conversion and assignment

See Also

Section 2.3.3 Section 2.3.3.6

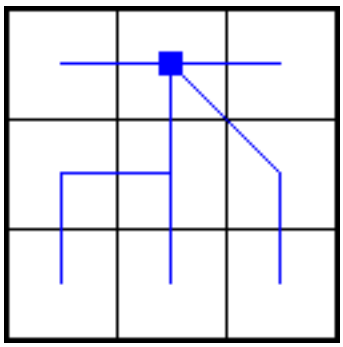
Examples

1.
```
pcrcalc  Result1.map = directional(Expr.map)
```

Result1.map

90	-1	270
90	0	315
0	0	0

Expr.map



2.
```
pcrcalc  --degrees Result2.map = directional(Expr.map)
```

Result2.map

350	0	0.01
MV	350	21
359	40	0

Expr.map

-10	0	0.01
MV	350	21
-1	400	0

downstream

Name

downstream -- Cell gets value of the neighbouring downstream cell

Synopsis

```
pcrcalc Result = downstream( ldd, expression )
```

`ldd`	ldd spatial
`expression`	boolean, nominal, ordinal, scalar, directional, ldd spatial
`Result`	type of `expression` spatial

Operation

For each cell, assigns to `Result` the `expression` value of the neighbouring downstream cell, where downstream cells are determined using the local drain directions on `ldd`. In case a cell doesn't have a downstream cell (i.e. a pit) its own `expression` value is assigned to `Result`.

Notes

A cell with a missing value on `ldd` and/or `expression` is assigned a missing value on `Result`. Its upstream neighbours are assigned a missing value also.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

lddmask

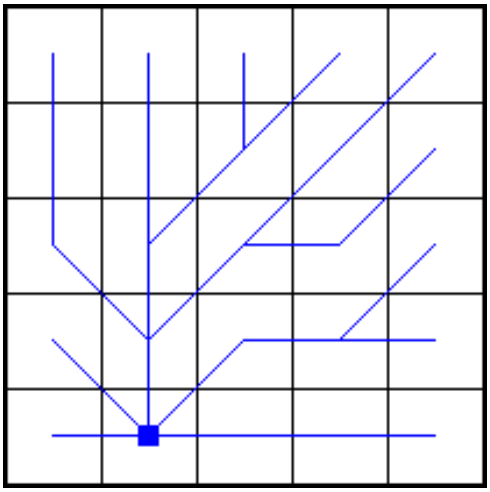
Examples

1.
```
pcrcalc   Result.map=downstream( Ldd.map, Expr.map)
```

Result.map

1	1	2	2	2
2	2	2	MV	4
2	2	MV	MV	4
7	7	7	2	4
7	7	7	5	5

Ldd.map



Expr.map

1	1	2	2	4
1	1	2	2	4
2	2	MV	4	4
2	2	2	4	4
3	7	5	5	5

Prev directional
Home Up
Next downstreamdist

downstreamdist

Name

downstreamdist -- Distance to the first cell downstream

Synopsis

```
pcrcalc [option] Result = downstreamdist( ldd )
```

`ldd`	ldd spatial
`Result`	scalar spatial

Options

--unittrue or --unitcell

--unittrue

distance is measured in true distance (default)

--unitcell

distance is measured in number of cell lengths

Operation

For each cell, assigns to `Result` the distance to the first cell downstream, where downstream cells are determined using the local drain directions on `ldd`. This distance is the length of one cell in case the local drain direction is to one of the right, left, top or bottom neighbouring cells or sqrt(2) multiplied by the length of one cell in case the local drain direction is to one of the 4 neighbouring cells in diagonal directions. In case a cell doesn't have a downstream cell (i.e. a pit) a zero is assigned to `Result`.

Notes

A cell with a missing value on `ldd` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

lddmask

Examples

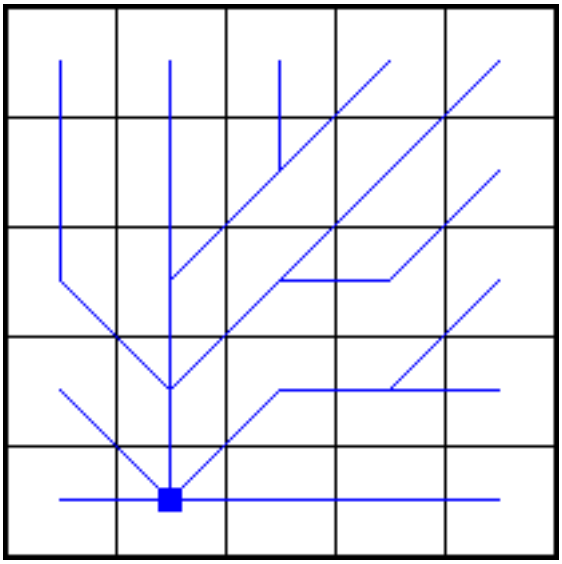
1.

```
pcrcalc  Result.map = downstreamdist(Ldd2.map)
```

`Result.map`

2	2	2	2.83	2.83
2	2	2.83	2.83	2.83
2.83	2	2.83	2	2.83
2.83	2	2.83	2	2
2	0	2	2	2

`Ldd2.map`



eq or ==

Name

eq or == -- Relational-equal-to operation on two expressions

Synopsis

pcrcalc Result = expression1 **eq** expression2
pcrcalc Result = expression1 **==** expression2

expression1	boolean, nominal, ordinal, scalar, directional, ldd spatial, non spatial
expression2	type of expression1 spatial, non spatial
Result	boolean spatial; non spatial if expression1 and expression2 are non spatial

Operation

For each cell evaluates expression1 in relation to expression2. If the cell value on expression1 equals the value on expression2 Result has a cell value 1 (condition is TRUE) on the corresponding cell; if the cell value on expression1 does not equal the value on expression2 Result has a cell value 0 (condition is FALSE).

Notes

A cell with a missing value on expression1 and/or expression2 or is assigned a missing value on Result.

The == sign is an alternative notation for **eq**.

Group

This operation belongs to the group of Comparison operators

Examples

1.

```
pcrcalc  Result.map =  Expr1.map eq Expr2.map
```

Result.map

0	0	0
0	0	1
0	MV	1

Expr1.map

4	11	-4
-4	2	3.8
5	MV	0

Expr2.map

2	-11	-4.1
0	4	3.8
-8	2	0

---

exp

Name

exp -- $Base_e$ exponential

Synopsis

pcrcalc Result = **exp**(power)

power	scalar
	spatial, non spatial
Result	scalar
	dimension of power

Operation

For each cell, raises e to the *n*th power, where *n* is the cell value on power. The result of this calculation is assigned to the corresponding cell on Result.

Notes

A cell a with missing value on power is assigned a missing value on Result.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.
```
pcrcalc  Result.map = exp(Power.map)
```

exp

Result.map

7.39	6.05	0.368
1	0.247	MV
2.01	0.0498	2.72

Power.map

2	1.8	-1
0	-1.4	MV
0.7	-3	1

Prev eq or ==
Home
Up
Next fac

fac

Name

fac -- Faculty or factorial of a natural positive number

Synopsis

```
pcrcalc Result = fac( expression )
```

`expression`	scalar
	spatial
`Result`	scalar
	dimension of `expression`

Operation

Computes the faculty or factorial of a natural positive number. Domain errors result in a MV. Overflow occurs on 35 and bigger, resulting in a MV.

ge or >=

Name

ge or >= -- Relational-greater-than-or-equal-to operation

Synopsis

```
pcrcalc Result = expression1 ge expression2
pcrcalc Result = expression1 >= expression2
```

`expression1`	ordinal, scalar
	spatial, non spatial
`expression2`	type of `expression1`
	spatial, non spatial
`Result`	boolean
	spatial; non spatial if `expression1` and `expression2` are non spatial

Operation

For each cell evaluates `expression1` in relation to `expression2`. If the cell value on `expression1` is greater than or equal to the value on `expression2` `Result` has a cell value 1 (condition is TRUE) on the corresponding cell; if the cell value on `expression1` is less than the value on `expression2` `Result` has a cell value 0 (condition is FALSE).

Notes

A cell with missing value on `expression1` and/or `expression2` results in a missing value on `Result` at the corresponding cell. The `>=` sign is an alternative notation for **ge**.

Group

This operation belongs to the group of Comparison operators

Examples

1.

```
pcrcalc  Result.map = Expr1.map ge Expr2.map
```

Result.map		
1	1	1
0	0	1
1	MV	1

Expr1.map		
4	11	-4
-4	2	3.8
5	MV	0

Expr2.map		
2	-11	-4.1
0	4	3.8
-8	2	0

Prev fac | Home | Up | Next gt or >

gt or >

Name

gt or > -- Relational-greater-than operation

Synopsis

```
pcrcalc Result = expression1 gt expression2
pcrcalc Result = expression1 > expression2
```

`expression1`	ordinal, scalar spatial, non spatial
`expression2`	type of `expression1` spatial, non spatial
`Result`	boolean spatial; non spatial if `expression1` and `expression2` are non spatial

Operation

For each cell evaluates `expression1` in relation to `expression2`. If the cell value on `expression1` is greater than the value on `expression2` `Result` has a cell value 1 (condition is TRUE) on the corresponding cell; if the cell value on `expression1` equals or is less than the value on `expression2` `Result` has a cell value 0 (condition is FALSE).

Notes

A cell with missing value on `expression1` and/or `expression2` results in a missing value on `Result` at the corresponding cell. The > sign is a alternative notation for **gt**.

Group

This operation belongs to the group of Comparison operators

Examples

1.

```
pcrcalc  Result.map = Expr1.map gt Expr2.map
```

Result.map

1	1	1
0	0	0
1	MV	0

Expr1.map

4	11	-4
-4	2	3.8
5	MV	0

Expr2.map

2	-11	-4.1
0	4	3.8
-8	2	0

---

idiv

Name

idiv -- Quotient of integer division of values on first expression by values on second expression

Synopsis

pcrcalc Result = expression1 **idiv** expression2

expression1	scalar
	spatial, non spatial
expression2	scalar
	spatial, non spatial
Result	scalar
	spatial; non spatial if expression1 and expression2 are non spatial

Operation

For each cell, the value on expression1 is divided (integer division) by the value on expression2. This quotient is assigned to the corresponding cell on Result.

Notes

A cell with 0 on expression2 is assigned a missing value on Result. A cell with missing value on expression1 and/or expression2 is assigned a missing value on Result.

Group

This operation belongs to the group of Arithmetic operators

See Also

/ or div mod

Examples

1.

```
pcrcalc  Result.map = Expr1.map idiv Expr2.map
```

Result.map		
MV	3	-3
0	3	3
2	2	8

Expr1.map		
3.5	7	-7
0	9.6	9.6
5.2	5.2	56

Expr2.map		
0	2	2
7	3	-3
2	-2	7

Prev gt or >
Home
Up
Next if then

if then

Name

if then -- Boolean condition determining whether value of expression or missing value is assigned to result

Synopsis

```
pcrcalc Result = if( condition then expression )
pcrcalcResult = if( condition, expression )
```

`condition`	boolean
	spatial, non spatial
`expression`	boolean, nominal, ordinal, scalar, directional, ldd
	spatial, non spatial
`Result`	type of `expression`
	spatial; if `condition` and `expression` are non spatial non spatial

Operation

The cell values on `condition` are interpreted as Boolean values where 1 is TRUE and 0 is FALSE. For each cell, the cell value on `condition` determines whether the value of the corresponding cell on `expression` or a missing value is assigned to the corresponding cell on `Result`: if `condition` has a cell value 1 (TRUE) the value on `expression` is assigned to `Result`, if `condition` has a cell value 0 (FALSE) a missing value is assigned to `Result`.

Notes

A cell with missing value on `condition` and/or `expression` results in a missing value on `Result` at the corresponding cell. A comma between `condition` and `expression` in the command line is an alternative notation for **then**.

If you want to cut an local drain direction map (data type ldd), use the operator **lddmask** instead of **if then**. The operator **if then** allows for cutting an `expression` of data type ldd, but we advice to use it in very special cases only; it will result in an unsound ldd.

Group

This operation belongs to the group of Conditional operators

See Also

cover defined lddmask

Examples

1.

```
pcrcalc  Result.map = if(Cond.map then Expr1.map)
```

Result.map

2	0	MV
4	MV	-6
MV	MV	3

Cond.map

1	1	0
1	0	1
0	MV	1

Expr1.map

2	0	3
4	MV	-6
-7	1	3

Prev idiv | Home Up | Next if then else

if then else

Name

if then else -- Boolean condition determining whether value of the first or second expression is assigned to result

Synopsis

```
pcrcalc Result = if( condition then expression1 else expression2 )
pcrcalcResult = if( condition, expression1, expression2 )
```

`condition`	boolean
	spatial, non spatial
`expression1`	boolean, nominal, ordinal, scalar, directional, ldd
	spatial, non spatial
`expression2`	type of `expression1`
	spatial, non spatial
`Result`	type of `expression1`
	spatial; non spatial if `condition`, `expression1` and `expression2` are all non spatial

Operation

The cell values on `condition` are interpreted as Boolean values where 1 is TRUE and 0 is FALSE. For each cell, the cell value on `condition` determines whether the value of the corresponding cell on `expression1` or `expression2` is assigned to the corresponding cell on `Result`: if condition has a cell value 1 (TRUE) the value on `expression1` is assigned `Result`, if `condition` has a cell value 0 (FALSE) the value on `expression2` is assigned to `Result`.

Notes

A cell with a missing value on `condition` results in a missing value on `Result` at the corresponding cell. A cell with a value 1 (TRUE) on `condition` and a missing value on `expression1` results in a missing value on `Result`. Also, a cell with a value 0 (FALSE) on `condition` and a missing value on `expression2` results in a missing value on `Result`.

Remember that the data type of `expression1` must correspond with the data type of `expression2`.

Comma's between `condition`, `expression1` and `expression2` in the command line is an alternative notation for **then else**.

If you want to cut an local drain direction map (data type ldd), use the operator **lddmask** instead of **if then else**. The operator **if then else** allows for cutting expressions (`expression1`, `expression2`) of data type ldd but we advice to use it in very special cases only; it will result in an unsound ldd.

Group

This operation belongs to the group of Conditional operators

See Also

cover defined lddmask

Examples

1.

```
pcrcalc  Result.map = if(Cond.map,Expr1.map,Expr2.map)
```

Result.map

2	0	44
4	47	-6
MV	MV	3

Cond.map

1	1	0
1	0	1
0	MV	1

Expr1.map

2	0	3
4	MV	-6
-7	1	3

Expr2.map

48	43	44
41	47	49
MV	MV	MV

kinematic

Name

kinematic -- Dynamic calculation of streamflow through a channel

Synopsis

pcrcalc QnewResult = **kinematic**(ldd, Qold, q, alpha, beta, dT, dX)

ldd	ldd
	spatial
Qold	scalar
	spatial, non spatial
q	scalar
	spatial, non spatial
alpha	scalar
	spatial, non spatial
beta	scalar
	spatial, non spatial
dT	scalar
	spatial, non spatial
dX	scalar
	spatial, non spatial
QnewResult	scalar
	spatial

Operation

The objective of this operator is to solve the kinematic wave.

The kinematic wave equations are [chow88]:

{{ delta Q } over { delta x }} +
{{ delta A } over { delta t }} = q

and

A = alpha { Q sup { beta }}

combined into

{{ delta Q } over { delta x }} +
{{ alpha } { beta } { Q sup { beta -1 }}
{{ delta Q } over { delta t }} = q }

with

Q streamflow through channel (m3/sec)
q inflow into the channel (m3/sec)
dt timestep used in the model (sec)
dx channel length through cell (m)
alpha coefficient
beta coefficient

The objective is to solve the equations for

Q tsub {t+1}

(Qnew) at each point in the map, given the channel parameters

alpha

and

beta

, the lateral inflow q and the initial conditions Qold. For each cell calculates the accumulated amount of material that flows out of the cell into its neighbouring downstream cell. This accumulated amount is the amount of material in the cell itself plus the amount of material in upstream cells of the cell.

Unlike the accuflux-family of functions this function allows for streamflow calculations in those situations where the average travel time through a cell is within the magnitude of the model time step.

Notes

This is an experimental implementation of the numerical solutions of the kinematic wave equations. The stability and accuracy of the equations can not be gauranteed under all circumstances.

The kinematic-operations uses (unlike other pcraster commands) defined units for the calculations. Whereas other pcraster commands can operate with any consistent system of units, the kinematic-operation explicitly needs its input parameters in (cubic) meters and seconds, and the result is in qubic meters per second. This explains the explicit use of dX and dT in the interface of the operator.

Additional sediment flux based on the channel flow calculated with this operator can be obtained by the code shown in example 1 below.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

Section 4.4.5

Examples

1.

```
pcrcalc  Qnew = kinematic(ldd.map, Qold, q,alpha, beta,dT,dX);
Qin  = upstream(ldd.map,Qnew);
Qsum = Qin+Qold+q;
Snew = accufractionflux(ldd.map,Sold+s,Qnew/Qsum);
```

Result.map

1	0	1
MV	1	1
1	1	0

Expr.map

-10	0	0.01
MV	350	21
-1	400	0

ldd

Name

ldd -- Data conversion from specific data types to local drain direction data type

Synopsis

pcrcalc [option] Result = **ldd**(expression)

expression	nominal, ordinal, directional
	spatial, non spatial
Result	ldd
	spatial, non spatial

Options

conversion from directional data type: --degrees or --radians

--degrees

values on expression are interpreted as degrees (default)

--radians

values on expression are interpreted as radians

Operation

If the expression is a PCRaster map or a calculation resulting in a PCRaster map, it is converted. If the expression has a data type nominal or ordinal, only the data type is changed; the cell values on Result correspond with the values on expression. For this conversion it is required that the cell values (or directions) on expression are in the domain of the ldd data type, i.e. a whole number from 1 up to and including 9. The values resemble the layout of the numeric key pad of your computer. If expression has a directional data type, the circular directional scale of expression is converted to the local drain direction codes of the ldd data type as follows: the local drain direction codes are interpreted as real clockwise directions where a local drain direction to the top of the map (ldd code 8) is 0 degrees. Each directional cell value on expression is assigned the ldd code of the local drain direction which is closest to the direction given by expression. For instance, assuming the option --degrees is set, all expression values equal to 22.5 or between 22.5 and 67.5 (i.e values in the range

[22.5,67.5>) are assigned a ldd code 9 on `Result`.
Or it generates a map of local drain direction data type with one constant value.

If `expression` has no PCRaster data type, a `Result` with data type ldd is generated. This is the case if `expression` is a number. The value of `expression` must be in the domain of the ldd data type, i.e. a whole number from 1 up to and including 9. `Result` will be a map with the same location attributes as the global clone map; all cells will have the value of `expression`.

Notes

A missing value on `expression` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Conversion and assignment

See Also

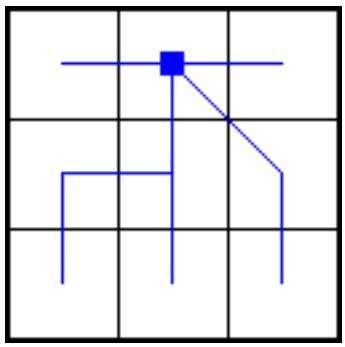
Section 2.3.3 Section 2.3.3.7

Examples

1.

```
pcrcalc  Result.map = ldd(Expr.map)
```

`Result.map`



`Expr.map`

100	-1	250
67.5	358	310
359	2	4

lddcreate

Name

lddcreate -- Local drain direction map with flow directions from each cell to its steepest downslope neighbour

Synopsis

pcrcalc [option] Result = **lddcreate**(elevation, outflowdepth, corevolume, corearea, catchmentprecipitation)

elevation	scalar
	spatial
outflowdepth	scalar
	spatial, non spatial
corevolume	scalar
	spatial, non spatial
corearea	scalar
	spatial, non spatial
catchmentprecipitation	scalar
	spatial, non spatial
Result	ldd
	spatial

Options

pit removing at edges of the map

--lddout

small catchments at the edge of the map are not considered as potentially being affected by the pit removing process: pits which are at the edge of the map are not removed. These pits remain in the map as outflow points of these small catchments (default).

--lddin

pits at the edge of the map (outflow points of a catchment) are removed like the other pits (if they have core dimensions smaller than the pit dimension thresholds). On the result their original catchment cells (including the pit cell) will drain in another catchment.

--unittrue (default) or --unitcell

--unittrue

elevation, outflowdepth and catchmentprecipitation is measured in true length, corearea in true area and corevolume in true volume. Units used for elevation and horizontal distance in x and y direction must be the same (default).

--unitcell

`elevation`, `outflowdepth` and `catchmentprecipitation` is measured in number of cell lengths, `corearea` in number of cells and `corevolume` in number of 3D blocks with edges of one cell length.

Operation

The operator creates a local drain direction map using the 8 point pour algorithm with flow directions from each cell to its steepest downslope neighbour. It determines for each cell its neighbour cell to whom material (for example water) will flow to. Each cell on the local drain direction map `Result` is assigned an arrow pointing to this downstream cell. This is the local drain direction of the cell. These directions linked to each other results in a local drain direction network: the flow pattern on the map. The directions are coded according to the standard codes used for the local drain direction data type:

Figure 1. Direction of ldd values. A value 5 (not shown) defines a cell without a local drain direction (a pit)

```
this "figures/lddcode.eps" figure placeholder
```

Coincidently, the values resemble the layout of the numeric key pad of your computer. A code 5 represents a pit, which is a cell without a local drain direction; it is surrounded by cells draining towards the cell. Additionally each catchment on `Result` ends with a pit cell at the edge of the map, this cell is considered to be the outflow point of the catchment.

For each cell, the local drain direction is determined on basis of the elevation cell values on the digital elevation model `elevation`, in a 3 x 3 cells window with the cell under consideration in the centre. Dependent on the elevation of the centre cell with respect to its 8 surrounding cells, the local drain direction is determined as follows: if a cell has one or more neighbours with a lower elevation on `elevation` it is assigned the local drain direction to the neighbour cell which results in the steepest drainage slope. If two or more neighbouring cells with the steepest drainage slope can be found, the local drain direction is randomly chosen, to one of these cells.

A cell may have neighbours which are at the same elevation as the cell under consideration. One cell of this kind or a set of neighbouring cells of this kind represent a flat area. Two types of flat areas may occur. The first type is bordered on one or more sides by one cell or a set of cells at lower elevation. Flat areas of this type are filled in with local drain directions iteratively, starting at the edge of the flat area bordered by an area at a lower elevation: each time the local drain direction is determined for one of the cells on the flat area which has one or more neighbour(s) with a local drain direction not pointing back into the cell under consideration. The local drain direction of the cell under consideration will be in direction of one of these cells (randomly chosen).

The second type of flat area is a flat area surrounded by cells at higher elevation. Flat areas of this type are iteratively filled in with local drain directions starting at the edge of the flat area and assigning drain directions to cells which have neighbours draining into the cell under consideration. Each iteration, a cell is assigned a drain direction to one of its neighbouring cells on the flat area. The result of resolving flat areas of this type is a map in which all cells are assigned a drain direction, except one cell which is a pit.

Pits are defined as those cells that only have neighbours at higher elevation than the cell under consideration, or a cell somewhere in the centre of a flat area which is surrounded by cells at higher elevation, as foresaid. Therefore, pits are those cells that only have neighbours pointing towards them, and no neighbours at lower or equal elevation that they can point to.

Figure 2. A sprinkling of the few definitions used in theory of pit removing.

```
this "figures/demldd.eps" figure placeholder
```

Pits can be removed by assigning artificial local drain directions to depressions which do not have an outlet, see Figure. The catchment of a pit is circumscribed by the divide; this is a line which draws a boundary between cells that drain to the pit and cells that drain to one of the neighbouring catchments. For a pit which will be removed, the upstream path from the pit over the local drain directions towards and over the divide is determined which crosses the divide moving through cells at the lowest possible elevation. The cell on this path at the highest elevation is called the outflow cell, its elevation corresponds with the overflow level of the catchment of the pit. Now, the pit is removed by reversing the original local drain directions on this upstream path to the outflow cell. Water for instance which falls in a certain cell in the catchment is removed from the catchment by tracing the local drain directions as follows: first it moves over the original local drain directions towards the pit; then it leaves the catchment by following the traversed local drain direction path upstream to the

outflow cell and then downstream into the neighbouring catchment.

The choice whether a pit must be removed or not depends on by what a pit is caused by. Most of the pits will be due to data errors in the digital elevation model `elevation`, grid mismatch or lack of resolution. These artificial pits must be removed. Additionally pits can be caused by natural phenomena: they can occur at the lowest point in depressions in a landscape. For a proper analysis it is important to maintain these natural pits as real pits on the local drain direction map. The choice whether a pit is removed or not can be made dependent on the dimension of the core of the pit. The core of a pit is the zone which contains all cells in the pit catchment with an elevation lower than the outflow level, see Figure. The dimensions of a pit core can be defined by core depth, core volume, core area and catchmentprecipitation. The core depth of a pit is the overflow level of a pit minus the elevation of the pit cell, see Figure. The core volume of a pit is defined as the total volume that is needed to fill up the core area to the overflow level. The core area of a pit is the area of the cells in a core. The catchment precipitation is the amount of rainfall in a pit catchment needed for filling up a pit core, assuming that all water which falls in the pit catchment accumulates in the core. It is defined by the volume of a core divided by the area of the catchment of a core. Using these pit core dimensions the choice is made whether a pit is resolved or not: for each pit cell the core depth, core volume, core area and catchmentprecipitation in millimetres are calculated. If all of these values are smaller than the threshold values on respectively `outflowdepth`, `corevolume`, `corearea`,`catchmentprecipitation` at the pit cell under consideration, the pit will be removed; if one value is larger or equal it will not be removed.

In general artificial pits will be relatively small compared to pits which are considered to be natural features of the landscape. So, the pit core dimension thresholds permit for distinguishing between these two sorts of pits. Additionally some pits of a certain size can be removed from your map while keeping other pits of the same size. This can be done by choosing dimension threshold values which are different for each pit on your map. Try different values and use the PCRaster operators interactively by running **lddcreate** with different combinations of `outflowdepth`, `corevolume`,`corearea`,`catchmentprecipitation` each time adjusting the thresholds.

Notes

A cell with missing value on one or more of the input expressions is totally ignored during operation of **lddcreate**; it is assigned a missing value on `Result`.

Here, a somewhat generalized description of pit removing is given. For a detailed description, see Van Deursen, 1995.

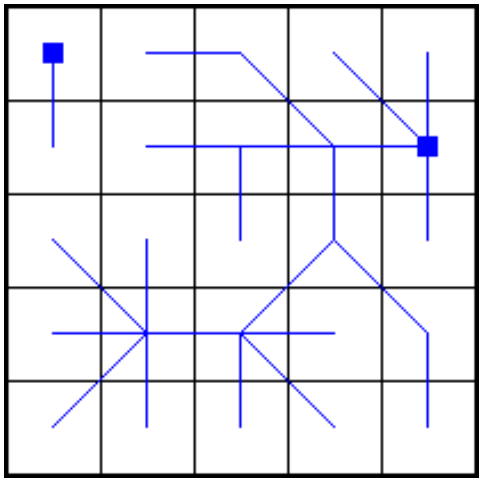
Group

This operation belongs to the group of Derivatives of elevation maps

Examples

1.

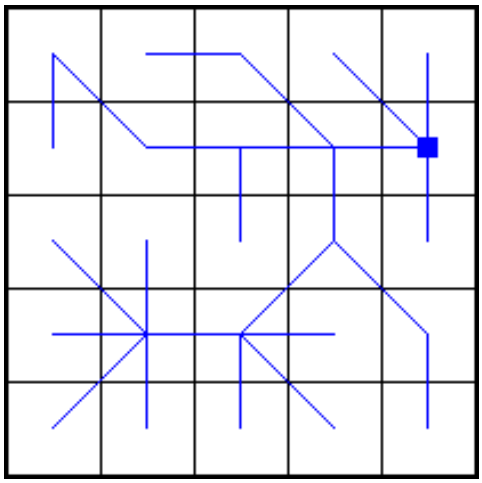
```
pcrcalc  Result1.map = lddcreate(Dem.map,1E35,1E35,1E35,1E35)
 Result1.map              Dem.map
```

2. 

23	29	11	7	6
35	24	7	5	3
36	21	13	12	17
37	10	10	15	19
35	31	27	24	29

```
pcrcalc  --lddin Result2.map = lddcreate(Dem.map,1E35,1E35,1E35,1E35)
```

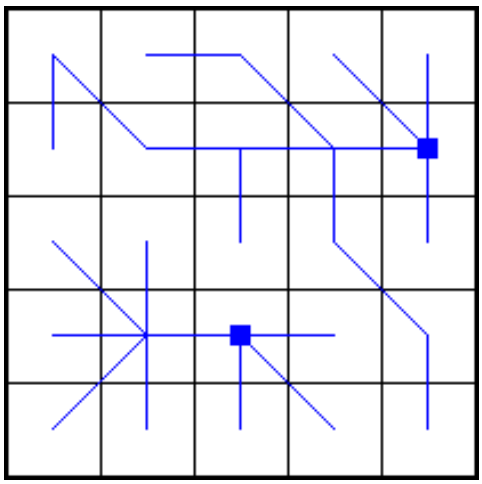
Result2.map Dem.map

3. 

23	29	11	7	6
35	24	7	5	3
36	21	13	12	17
37	10	10	15	19
35	31	27	24	29

```
pcrcalc  --lddin Result3.map = lddcreate(Dem.map,1E35,5000,1E35,1E35)
```

Result3.map Dem.map



23	29	11	7	6
35	24	7	5	3
36	21	13	12	17
37	10	10	15	19
35	31	27	24	29

lddcreatedem

Name

lddcreatedem -- Modified digital elevation model

Synopsis

pcrcalc [option] Result = **lddcreatedem**(elevation, outflowdepth, corevolume, corearea, catchmentprecipitation)

elevation	scalar
	spatial
outflowdepth	scalar
	spatial, non spatial
corevolume	scalar
	spatial, non spatial
corearea	scalar
	spatial, non spatial
catchmentprecipitation	scalar
	spatial, non spatial
Result	scalar
	spatial

Options

pit removing at edges of the map

--lddout

small catchments at the edge of the map are not considered as potentially being affected by the pit removing process: pits which are at the edge of the map are not removed. These pits remain in the map as outflow points of these small catchments (default).

--lddin

pits at the edge of the map (outflow points of a catchment) are removed like the other pits (if they have core dimensions smaller than the pit dimension thresholds). On the result their original catchment cells (including the pit cell) will drain in another catchment.

--unittrue (default) or --unitcell

--unittrue

`elevation`, `outflowdepth` and `catchmentprecipitation` is measured in true length, `corearea` in true area and `corevolume` in true volume. Units used for elevation and horizontal distance in x and y direction must be the same (default).

--unitcell

`elevation`, `outflowdepth` and `catchmentprecipitation` is measured in number of cell lengths, `corearea` in number of cells and `corevolume` in number of 3D blocks with edges of one cell length.

assignment of elevation in pits

--lddfill

for each pit which is removed, cells in the area which was formerly the core of the pit are assigned an elevation equal to the overflow level of the pit core (default).

--lddcut

for each pit which is removed, cells on the path between the pit and the outflow cell are assigned the elevation of the pit cell. The elevation of the other cells in the core of the pit is not changed.

Operation

This operation corresponds with the local drain direction maker **lddcreate** with the difference that a modified digital elevation model is created instead of a local drain direction map. The modified digital elevation model fits the local drain direction map generated on the basis of the original digital elevation model. 'Not real' cores are removed from the local drain direction map. Additionally an extra option needed for creation of the modified digital elevation model can be specified.

The expressions used for the pit removing process `outflowdepth`, `corevolume`, `corearea`, `catchmentprecipitation` and the options - -unitcell/unittrue and --lddout/--lddin have exactly the same meaning and are used in the same way as with the **lddcreate** operation. So, before you start making modified dem's using **lddcreatedem** we advise you to read and study the description of the **lddcreate** command first.

First, a local drain direction map is generated internally, using `elevation`. This is done after the manner of the **lddcreate** operator, but it is not saved as an expression. Second, the original digital elevation model `elevation` is modified in such a way that it fits this local drain direction map. This modified digital elevation model is saved as `Result`. The cell values on `Result` correspond with the values on `elevation`, with the exception that the elevation of cells in cores of pit cells is changed. This is done for cores of pit cells which are removed only; the elevation in cores of pits which are not removed remains unaffected. The way elevation values in cores are changed is specified with the --lddfill and --lddcut options. Setting the option --lddfill the elevation of cells in a core is increased until the overflow level is reached. This can be compared with fluviatile or lacustrine sedimentation in the core depression until a maximum sedimentation level is reached: the level of the core pass which is at the lowest elevation. The option --lddcut does not fill the core but reduces the elevation of the cells on the path between the pit cell and the overflow cell. This can be compared with digging a canal in the core between the pit and the pass with the lowest elevation; the canal bottom is at the elevation of the pit.

Notes

A cell with missing value on one or more of the input expressions is totally ignored during operation of **lddcreatedem**; it is assigned a missing value on `Result`.

Here, a somewhat generalized description of pit removing and reversal of local drain directions is given. For a detailed description see Van Deursen, 1995.

Group

This operation belongs to the group of Derivatives of elevation maps

Examples

1.

```
pcrcalc  Result1.map = lddcreatedem(Dem.map,1E35,1E35,1E35,1E35)
```

Result1.map

23	29	11	7	6
35	24	7	5	3
36	21	13	12	17
37	12	12	15	19
35	31	27	24	29

Dem.map

23	29	11	7	6
35	24	7	5	3
36	21	13	12	17
37	10	10	15	19
35	31	27	24	29

2.

```
pcrcalc  --lddcut Result2.map = lddcreate(Dem.map,1E35,1E35,1E35,1E35)
```

Result2.map

23	29	11	7	6
35	24	7	5	3
36	21	13	10	17
37	10	10	15	19
35	31	27	24	29

Dem.map

23	29	11	7	6
35	24	7	5	3
36	21	13	12	17
37	10	10	15	19
35	31	27	24	29

lddist

Name

lddist -- Friction-distance from the cell under consideration to downstream nearest TRUE cell

Synopsis

pcrcalc [option] Result = **lddist**(ldd, points, friction)

ldd	ldd
	spatial
points	boolean
	spatial
friction	scalar
	spatial, non spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell

--unittrue

distance is measured in true distance (default)

--unitcell

distance is measured in number of cell lengths

Operation

points Is a Boolean expression where 0 is FALSE and 1 is TRUE. For each cell, the friction-distance over a friction surface is calculated of the path over the local drain direction network on ldd to the first downstream cell which is TRUE on points.

For each cell, the path starts at the centre of the cell itself. The path is a route over the consecutive neighbouring FALSE points cells in downstream

direction, where the direction is specified by the local drain directions on `ldd`. The friction-distance increases while following the path starting with a friction-distance of zero. The amount of increase per unit distance is specified by the values on `friction`. Using these values, increase of friction-distance when travelling from one cell to its first downstream cell is calculated as follows: Let `friction`(sourcecell) and `friction`(destinationcell) be the `friction` values at the cell where is moved from and at its first downstream cell where is moved to, respectively. While moving from the source cell to the destination cell the friction-distance increases with:

distance x {(`friction`(sourcecell)+`friction`(destinationcell)}/2

where distance is the distance between the centre of the sourcecell and the centre of the destination cell. This distance equals the cell length if the source cell and the destination cell are neighbours in horizontal or vertical directions; it equals sqrt(2) multiplied by the cell length if the cells are neighbours in diagonal directions.

For each cell its path is followed in downstream direction until a TRUE cell on `points` is reached. The friction-distance covered at the centre of this TRUE cell is assigned to the cell where the path started and saved as `Result`. If no cell is crossed which is TRUE on `points` a missing value is sent to the cell where the path started.

Notes

The values on `friction` must be larger than zero. For cells which are TRUE on `points` the friction-distance is zero and a 0 is assigned to `Result`. If a cell has a missing value on `ldd`, `points` and/or `friction`, a missing value is assigned to the corresponding cell and to all its upstream cells on `Result`.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

Section 4.4.4 slopelength Section 6.2.3

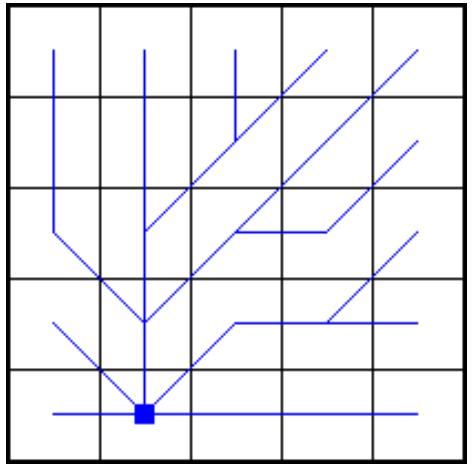
Examples

1.
```
pcrcalc  Result1.map = lddist(Ldd2.map,Points.map,1)
```

Result1.map

MV	6	6.83	7.66	8.49
MV	4	4.83	5.66	7.66
2.83	2	2.83	4.83	MV
MV	0	MV	MV	MV
MV	MV	MV	MV	MV

Ldd2.map



Points.map

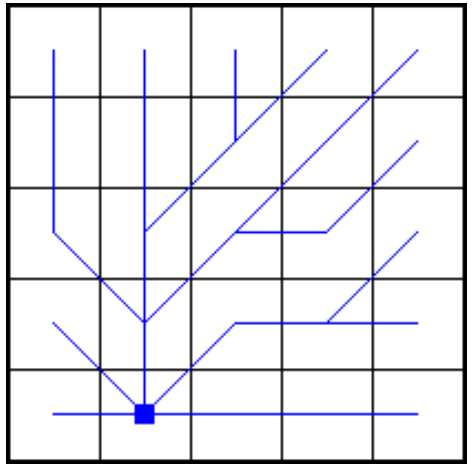
0	0	0	0	0
MV	0	0	0	0
0	0	0	0	0
0	1	0	0	0
0	0	0	0	0

2. `pcrcalc  Result2.map = lddist(Ldd2.map,Points2.map,FrictMat.map)`

Result2.map

12.5	MV	10	0	91.9
10.5	MV	0	28.3	31.4
8.49	10	14.1	24.1	MV
MV	0	MV	MV	0
MV	MV	0	6.9	13.8

Ldd2.map



Points2.map

0	0	0	1	0
0	0	1	0	0
0	0	0	0	0
0	1	0	0	1
0	0	1	0	0

FrictMat.map

1	5	5	5	40
1	MV	5	5	0.1
1	5	5	5	0.1
1	5	0.1	0.1	0.1
1	0.1	0.1	6.8	0.1

Prev
lddcreatedem

Home
Up

Next
lddmask

lddmask

Name

lddmask -- Local drain direction map cut into a (smaller) sound local drain direction map

Synopsis

```
pcrcalc Result = lddmask( ldd, mask )
```

`ldd`	ldd
	spatial
`mask`	boolean
	spatial
`Result`	ldd
	spatial

Operation

The cell values on `mask` are interpreted as boolean values, where 1 is TRUE and 0 is FALSE. The part of the local drain direction map `ldd` which you want to cut out must totally be filled with 1 (TRUE) values on `mask`.

Each cell with a `mask` value 0 (FALSE) is assigned a missing value on `Result`. Each cell with a `mask` value 1 (TRUE) is assigned a value which corresponds with the value on `ldd`, except cells with a `mask` value 1 that have a local drain direction on `ldd` towards a cell with a 0 (FALSE) on `mask`. These last named cells are outflow cells on the edge of the new ldd, these are assigned a cell value 5, which is the ldd code for a pit.

Notes

A cell with a missing value on `mask` is interpreted as a `mask` value 0 (FALSE) and handled in that way. In addition, a cell with a missing value on `ldd` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Missing value creation

See Also

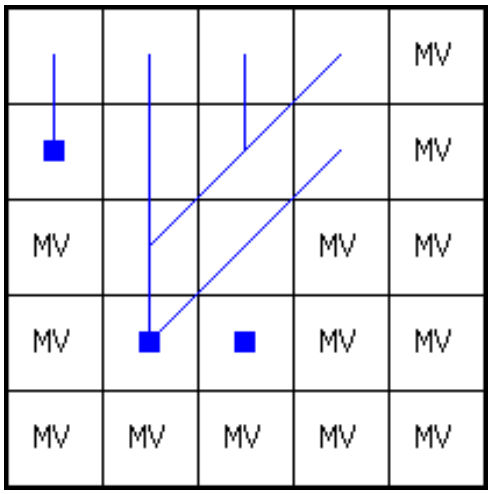
Section 6.2.4

Examples

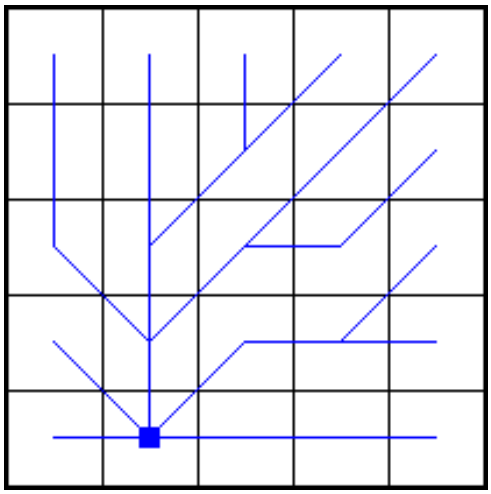
1.

```
pcrcalc  Result.map = lddmask(Ldd.map,Mask.map)
```

`Result.map`



`Ldd.map`



`Mask.map`

1	1	1	1	0
1	1	1	1	0
0	1	1	0	0
0	1	1	0	0
0	0	0	0	0

Prev lddist | Home | Up | Next lddrepair

lddrepair

Name

lddrepair -- Reparation of unsound local drain direction map

Synopsis

pcrcalc Result = **lddrepair**(ldd)

`ldd` ldd
spatial

Operation

Each cell on a local drain direction map must have a pit at the end of its downstream path. If this is not the case for one or more cells on a local drain direction map, the map is called unsound. An unsound local drain direction map can not be used as input expression for the operations with local drain direction maps.

The **lddrepair** operation changes the cell values on the unsound `ldd` in such a way that it becomes sound: all downstream paths will end in a pit cell; this adjusted `ldd` is saved as `Result`.

The repair operation is done as follows. Two things may be the cause of a downstream path not ending in a pit cell: a set of cells in a cycle and cells draining to a missing value or to the outside of the map. A cycle is a set of cells that don't drain to a pit because they drain to each other, in a closed cycle. The smallest cycle consists of two cells with local drain directions to each other; larger cycles may consist of several cells. First, the cycles are removed by assigning missing values to all cells in a cycle. Second, cells with a local drain direction to the outside of the map or to a cell with a missing value (including cells that were in a cycle) are assigned the ldd code of a pit cell (code: 5). Now, all downstream paths on the local drain direction map end in a pit cell; this adjusted map is saved as `Result`.

Notes

A missing value on `ldd` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Missing value creation

See Also

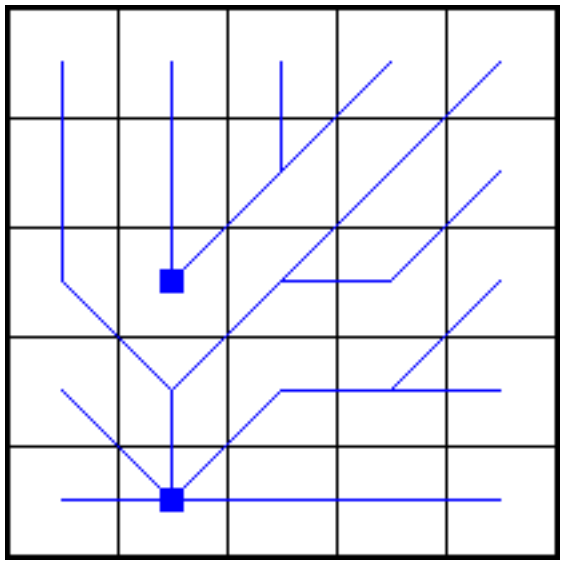
Section 6.2.4 Section 2.3.3.7

Examples

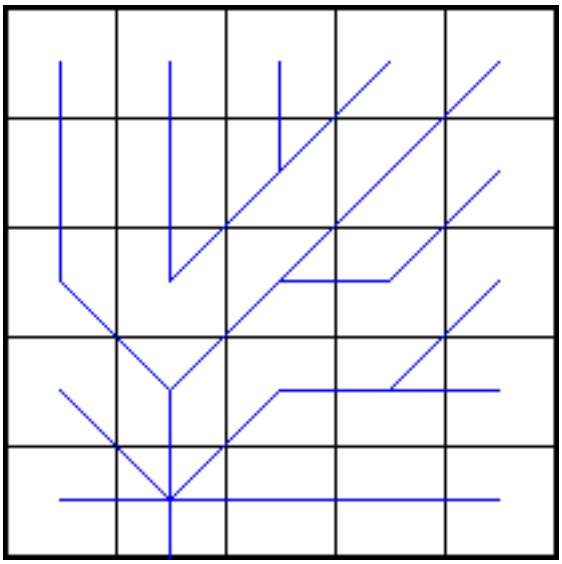
1.

```
pcrcalc  Result.map = lddrepair(Ldd.map)
```

Result.map



Ldd.map



Prev lddmask
Home Up
Next le or <=

le or <=

Name

le or <= -- Relational-less-than-or-equal-to operation

Synopsis

```
pcrcalc Result = expression1 le expression2
pcrcalc Result = expression1 <= expression2
```

`expression1`	ordinal, scalar
	spatial, non spatial
`expression2`	type of `expression1`
	spatial, non spatial
`Result`	boolean
	spatial; non spatial if `expression1` and `expression2` are non spatial

Operation

For each cell evaluates `expression1` in relation to `expression2`. If the cell value on `expression1` is less than or equal to the value on `expression2` `Result` has a cell value 1 (condition is TRUE) on the corresponding cell; if the cell value on `expression1` is greater than the value on `expression2` `Result` has a cell value 0 (condition is FALSE).

Notes

A cell with missing value on `expression1` and/or `expression2` results in a missing value on `Result` at the corresponding cell. The <= sign is an alternative notation for **le**.

Group

This operation belongs to the group of Comparison operators

Examples

1.

```
pcrcalc  Result.map =  Expr1.map le Expr2.map
```

Result.map

0	0	0
1	1	1
0	MV	1

Expr1.map

4	11	-4
-4	2	3.8
5	MV	0

Expr2.map

2	-11	-4.1
0	4	3.8
-8	2	0

---

ln

Name

ln -- Natural logarithm ($_e$)

Synopsis

pcrcalc Result = **ln**(expression)

expression	scalar
	spatial, non spatial
Result	scalar
	dimension of expression

Operation

For each cell, calculates the natural logarithm ($_e$) logarithm of the cell value on expression and assigns it to the corresponding cell on Result.

Notes

The cell values on expression must be greater than 0. Any cell value outside this range is assigned a missing value on Result. A cell with missing value on expression is assigned a missing value on Result.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.
```
pcrcalc  Result.map = ln(Expr.map)
```

Result.map

0	4.03	0.693
MV	0.833	2.64
MV	MV	-0.223

Expr.map

1	56	2
0	2.3	14
MV	-0.1	0.8

Prev le or <= | Home | Up | Next log10

log10

Name

log10 -- Log $_{10}$

Synopsis

pcrcalc Result = **log10**(expression)

expression	scalar
	spatial, non spatial
Result	scalar
	dimension of expression

Operation

For each cell, calculates the $_{10}$ logarithm of the cell value on `expression` and assigns it to the corresponding cell on `Result`.

Notes

The cell values on `expression` must be greater than 0. Any cell value outside this range is assigned a missing value on `Result`. A cell with missing value on `expression` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.
```
pcrcalc  Result.map = log10(Expr.map)
```

Result.map

MV	0	MV
MV	3	1.38
1.04	-0.886	-2

Expr.map

-10	1	MV
0	1e+03	24
11	0.13	0.01

Prev ln | Home | Up | Next lookup

lookup

Name

lookup -- Compares cell value(s) of one or more expression(s) with the search key in a table

Synopsis

```
pcrcalc [option] Result = lookupboolean( table, expression 1, expression 2,... expression n)
pcrcalc [option] Result = lookupnominal( table, expression 1, expression 2,... expression n)
pcrcalc [option] Result = lookupordinal( table, expression 1, expression 2,... expression n)
pcrcalc [option] Result = lookupscalar( table, expression 1, expression 2,... expression n)
pcrcalc [option] Result = lookupdirectional( table, expression 1, expression 2,... expression n)
pcrcalc [option] Result = lookupldd( table, expression 1, expression 2,... expression n)
```

`table`	ascii text table
`expression`1-n	boolean, nominal, ordinal, scalar, directional, ldd
	spatial
`Result`	type is specified by the sort of command: **lookupboolean** results in a boolean `Result`, **lookupnominal** results in a nominal `Result` etc.
	spatial; non spatial if all `expression`s 1-n are non spatial

Options

--columntable or --matrixtable

--columntable

a column table is read always (default)

--matrixtable

if two expressions `expression`1, `expression`2,...`expression`n are specified in the command line a matrix table is read instead of a column table.

--degrees or --radians

--degrees

directional values in `table` are interpreted as degrees (default)

--radians

directional values in `table` are interpreted as radians

cell representation for **lookupnominal** and **lookupordinal**

--small

cell representation is small integer cell representation (default)

--large

cell representation is large integer cell representation

Operation

operation with a column table (with global option --columntable):

In the `table` relations between the values of `expression`1, `expression`2,...`expression`n are given. For each combination of values of `expression`1, `expression`2,...`expression`n a new value can be specified.

If the option --columntable is set, a column table is always used for `table`. The column `table` consists of a number of *n* +1 columns. The first *n* columns are key columns, where *n* is the number of the one or more expressions `expression`1, `expression`2,...`expression`n. The key columns consist of key fields; each key field is one value or a ranges of values. The key fields in the first column will be linked to cell values on `expression`1, the key fields in the second column to values on `expression`2, and so on, where the key fields in the *n*th column will be linked to values on `expression`n. The last column (column number n+1) contains so called value fields; these are values which will be assigned to `Result`. Each row in the `table` is called a tuple. Of course, it consists of *n* key fields and one value field. An example of a column table is given in the Table below.

Example 1. Example of a column table. The first, second and third column give the values of *expression1*, *expression2* and *expression3* respectively; the fourth column contains the value fields.

<2,>	3	<,12>	1
<,2]	3	<,12>	3
<2,>	14	<,12>	7
<,2]	14	<,12>	9
<2,>	14	8	4
<,2]	14	8	8

For each cell, the **lookup** operator reads the values in that cell on `expression`1,`expression`2,...`expression`n and looks for a tuple whose key fields match these cell values, starting at the first row of the `table`. So, in a tuple the *i*th key field (where *i* is 1 to *n*) is compared with the value on `expression`i. It matches if the cell value on `expression`i is equal to the value in the key field or if it is in the range of the key field, in case of a key field consisting of a range of values. If all key fields in a tuple match the cell values of the expressions belonging to them, the value in the value field of the

tuple is sent to the corresponding cell on `Result`.

The `table` is an ordinary ascii text file which can be made using any text editor program you wish. Alternatively you can make it with a spread sheet program or word processing program and export `table` as an ascii text file. You can have a squint at your table by typing the DOS command **type** followed by a space and the table name `table`.

The precise format of the `table` is as follows. In a row (tuple), the number of key fields must equal the number of expressions `expression`1,`expression`2,...`expression`n. The key fields are followed by one value field. The fields must be separated by one or more spaces or tabs. The number of spaces or tabs doesn't matter. A value field is one single value. A key field is a single value, or a range of values, where a range of values is typed as: '[' or '<' symbol, minimum value, comma, maximum value, ']' or '>' symbol. The minimum and maximum values are included in the range if square brackets (respectively '[' and ']') are used, they are not included if '<' or '>' are used. Omitting a value in the range definition means infinity. Ranges can be used for nominal, ordinal, scalar and directional data types. Values in keys are typed as an ordinary number (for instance 24.453) or using 10 exponentials (for instance 32.45e3 means 32450). Examples of tuples are:

[,0.05> 1
Assign a 1 to all *expression* cell values smaller than 0.05

[-2e3,0> 2
Assign a 2 to all *expression* cell values equal to -200 or between -200 and 0.

[-1.42,-0.2> [,9> -2.2
Assign a -2.2 to all cells with: *expression1* cell value equal to -1.42 or between -1.42 and -0.2 and *expression2* cell value smaller than 9.

In one `table` as many tuples as needed can be specified. Remember that for each cell the value field is assigned of the *first* tuple (from top to bottom) that matches the set of `expression`1, `expression`2,...`expression`n values on the cell.

operation with a matrix table (with global option --matrixtable):

If two expressions `expression`1, `expression`2,...`expression`n are specified in the command line, `table` will be interpreted as a matrix table instead of a column table. If a different number of expressions `expression`1, `expression`2,...`expression`n is specified `table` will be read as a column table and the operation is performed as described above.

If a matrix table is read, the `table` must have the following make-up; Table 2 gives an example. The first field in the top left corner of the matrix, is not considered during operation but is necessary to align the matrix; it is a dummy field and may have any value. The first row consists of this dummy field and the key fields which will be linked to `expression`1. The first column consists of the dummy field and the key fields which will be linked to `expression`2. The key fields may be a single value or may be a range, where ranges are specified in the same way as it is done in a column table (see above). The remaining fields in the matrix are value fields and consist of the values which will be assigned to `Result`. In horizontal direction, fields must be separated by one or more spaces or tabs. All fields must be filled in.

Example 2. Example of a matrix table. The fields in the first row contain values of *expression1*; the fields in the first column contain values of *expression*. The field in the top left corner is a dummy field. The remaining fields are value fields.

```
-99   1     2     3     4
12   6.5   6.5    6     6
14   -4    -4    -4    -4
16  -13   -13   -12   -12
}}
```

For each cell, the operator reads the `expression`1 value and the `expression`2 value. It searches in the matrix both the column with a key field value which matches the `expression`1 value and the row with a key field value which matches the `expression`2 value. The value field which is in this column and row is assigned to the corresponding cell on `Result`. If more than one combination of column key field and row key field match the cell values of `expression`1 and `expression`2, the value field is chosen which is firstly in the most left column and secondly in the most upper row.

Notes

If a cell has `expression`1 and/or `expression`2 and/or...`expression`n values that don't match with a key field in the table, a missing value is assigned to the cell on `Result`.

A cell with a missing value on an expression `expression`1, `expression`2,...`expression`n is assigned a missing value on `Result`.

Each value in the key fields must be in the domain of the (sub) data type of the expression to which it will be linked, else the operation will give an error. Also, the value fields must be in the domain of the data type of the `Result` map (specified by the type of command: **lookupboolean**, **lookupnominal**,.. etc.).

Using **lookupldd** for generating a `Result` of data type ldd is quite risky: probably it will result in a ldd which is unsound. If you do want to create a `Result` of data type ldd use the operator **lddrepair** afterwards. This operator will modify the ldd in such a way that it will be sound, see the operator **lddrepair**.

Group

This operation belongs to the group of Point operators; relations in tables

See Also

lddrepair Section 3.3.2

Examples

1.

```
pcrcalc  Result1.map = lookupnominal(Table.txt,Expr1.map)
```

Result1.map

1	1	MV	3
5	5	7	MV
MV	9	9	11
MV	7	7	7

Table.txt

```
[, -2.5>                    1
 -2.5
<-2.5, 0]                   5
<0, 10>                     7
[12.5, 17.75]   9
<17.75, 250>    11
<0, 1>                      13
```

Expr1.map

3	-2.7	-12	MV	-2.5
	-2.49	0	3	10
	11.8	12.5	14.1	111
	312	0.5	0.4	1.2

2. `pcrcalc  Result2.map = lookupordinal(Table2.txt,Expr12.map,Expr22.map,Expr32.map)`

Result2.map

3	MV	1	1
3	1	1	1
6	5	4	4
5	5	5	MV

Table2.txt

```
1   7    <,>   1
1  <7,>  [0,>  2
1  <7,>  <,0>  3
0   7    <,>   4
0  <7,>  [0,>  5
0  <7,>  <,0>  6
```

Expr12.map

1	1	1	1
1	1	1	1
0	0	0	0
0	0	0	MV

Expr22.map

12	2	7	7
26	7	7	7
26	48	7	7
14	17	16	18

Expr32.map

-11	-4	-6	0
-0.5	-4	-3	3
-3	0	0.6	4.1
1	2	7	12

3. `pcrcalc  --matrixtable Result3.map = lookupordinal(Table2.txt,1,100,Expr32.map)`

Result3.map

3	3	3	2
3	3	3	2
3	2	2	2
2	2	2	2

Table2.txt

```
1   7    <,>   1
1  <7,>  [0,>  2
1  <7,>  <,0>  3
0   7    <,>   4
0  <7,>  [0,>  5
0  <7,>  <,0>  6
```

Expr32.map

-11	-4	-6	0
-0.5	-4	-3	3
-3	0	0.6	4.1
1	2	7	12

4. `pcrcalc  --matrixtable Result4.map = lookupscalar(Table3.txt,Expr13.map,Expr23.map)`

Result4.map

-12.5	-6	MV
-12.5	-6	-6
12.5	12.5	12.5

Table3.txt

```
-9 0     1
2  12.5 -12.5
7  13.5 -6
9  14   -2
```

Expr13.map

1	1	MV
1	1	1
0	0	0

Expr23.map

2	7	7
2	7	7
2	2	2

Prev
log10

Home
Up

Next
lt or <

lt or <

Name

lt or < -- Relational-less-than operation

Synopsis

pcrcalc `Result = expression1` **lt** `expression2`

pcrcalc `Result = expression1` **<** `expression2`

`expression1`	ordinal, scalar
	spatial, non spatial
`expression2`	type of `expression1`
	spatial, non spatial
`Result`	boolean
	spatial; non spatial if `expression1` and `expression2` are non spatial

Operation

For each cell evaluates `expression1` in relation to `expression2`. If the cell value on `expression1` is less than the value on `expression2` `Result` has a cell value 1 (condition is TRUE) on the corresponding cell; if the cell value on `expression1` equals or is greater than the value on `expression2` `Result` has a cell value 0 (condition is FALSE).

Notes

A cell with a missing value on `expression1` and/or `expression2` results in a missing value on `Result` at the corresponding cell.

The < sign is a alternative notation for **lt**.

Group

This operation belongs to the group of Comparison operators

Examples

1.

```
pcrcalc  Result.map = Expr1.map lt Expr2.map
```

Result.map		
0	0	0
1	1	0
0	MV	0

Expr1.map		
4	11	-4
-4	2	3.8
5	MV	0

Expr2.map		
2	-11	-4.1
0	4	3.8
-8	2	0

---

maparea

Name

maparea -- Total map area

Synopsis

pcrcalc [option] Result = **maparea**(expression)

expression	boolean, nominal, ordinal, scalar, directional, ldd
	spatial
Result	scalar
	non spatial

Options

--unittrue or --unitcell

--unittrue

area is computed in true area (default)

--unitcell

area is computed in number of cells

Operation

Sums the area of the non missing value cells on expression and assigns this total area to Result (non spatial).

Group

This operation belongs to the group of Map operators

Examples

1.

```
pcrcalc  Result.map= maparea(Expr.map)
```

Result.map

80	80	80
80	80	80
80	80	80

Expr.map

MV	MV	4
MV	0	-3
MV	7	1

Prev lt or <
Home
Up
Next mapmaximum

mapmaximum

Name

mapmaximum -- Maximum cell value

Synopsis

pcrcalc Result = **mapmaximum**(expression)

expression	ordinal, scalar
	spatial
Result	type of expression
	non spatial

Operation

Determines the maximum cell value of the expression cell values and assigns this value to Result (non spatial).

Notes

The value of Result is undefined if all cells of expression are missing value.

Group

This operation belongs to the group of Map operators

Examples

1.

```
pcrcalc  Result.map = mapmaximum(Expr.map)
```

Result.map

8	8	8
8	8	8
8	8	8

Expr.map

8	0	4
4	4	-3
MV	7	-9

Prev maparea | Home Up | Next mapminimum

mapminimum

Name

mapminimum -- Minimum cell value

Synopsis

pcrcalc `Result =` **mapminimum**`( expression )`

`expression`	ordinal, scalar
	spatial
`Result`	type of `expression`
	non spatial

Operation

Determines the minimum cell value of `expression` cell values and assigns this value to `Result` (non spatial).

Notes

The value of `Result` is undefined if all cells of `expression` are missing value.

Group

This operation belongs to the group of Map operators

Examples

1.
```
pcrcalc  Result.map = mapminimum(Expr.map)
```

Result.map

-9	-9	-9
-9	-9	-9
-9	-9	-9

Expr.map

8	0	4
4	4	-3
MV	7	-9

Prev mapmaximum | Home | Up | Next mapnormal

mapnormal

Name

mapnormal -- Cells get non spatial value taken from a normal distribution

Synopsis

pcrcalc Result = **mapnormal** ()

Result scalar
non spatial

Operation

A random generator is used to generate the Result: a value is taken from a normal distribution with mean 0 and standard deviation 1. This value is assigned to Result (non spatial).

Notes

Group

This operation belongs to the group of Random number generators; Maps

See Also

Section 6.1.11 Section 6.3.2

Examples

1.

```
pcrcalc  --clone Expr.map Result.map = mapnormal()
```

Result.map

-2.16	-2.16	-2.16
-2.16	-2.16	-2.16
-2.16	-2.16	-2.16

Expr.map

1	2	-6
5	4	3
2	8	MV

Prev mapminimum | Home | Up | Next maptotal

maptotal

Name

maptotal -- Sum of all cell values

Synopsis

```
pcrcalc Result = maptotal( expression )
```

`expression`	scalar
	spatial
`Result`	scalar
	non spatial

Operation

Sums all `expression` cell values and assigns this sum of these values to `Result` (non spatial).

Notes

The value of `Result` is not correct if all cells of `expression` are missing value. In that case `Result` is assigned the value 0.

Group

This operation belongs to the group of Map operators

Examples

1.
```
pcrcalc  Result.map = maptotal(Expr.map)
```

Result.map

15	15	15
15	15	15
15	15	15

Expr.map

8	0	4
4	4	-3
MV	7	-9

Prev mapnormal
Home Up
Next mapuniform

mapuniform

Name

mapuniform -- Cells get non spatial value taken from an uniform distribution

Synopsis

pcrcalc Result = **mapuniform** ()

Result scalar
non spatial

Operation

A random generator is used to generate the Result: a random number between 0 and 1 is taken from a uniform distribution. This value is assigned to Result (non spatial).

Notes

Group

This operation belongs to the group of Random number generators; Maps

See Also

Section 6.1.11 Section 6.3.2

Examples

1.
```
pcrcalc  --clone Expr.map Result.map = mapuniform()
```

Result.map

0.715	0.715	0.715
0.715	0.715	0.715
0.715	0.715	0.715

Expr.map

1	2	-6
5	4	3
2	8	MV

Prev maptotal | Home Up | Next max

max

Name

max -- Maximum value of multiple expressions

Synopsis

pcrcalc Result = **max**(expression 1, expression 2,... expression n)

expression1-n	ordinal, scalar; expression1, expression2,...expressionn must have the same data type
	spatial, non spatial
Result	type of expression1,expression2,...expressionn
	spatial; if all expression1, expression2,...expressionn are non spatial: non spatial

Operation

For each cell, the maximum value of expression1, expression2,...expressionn is determined and assigned to the corresponding cell on Result. As many expressions can be specified as needed.

Notes

A cell with missing value on one or more expressions results in a missing value on the corresponding cell on Result.

Group

This operation belongs to the group of Maximize, minimize

Examples

1.

```
pcrcalc  Result1.map =  max(Expr1.map,Expr2.map)
```

Result1.map

8	7	-2
14	1	0
-1	MV	MV

Expr1.map

8	6	-2
4	1	0
-7	8	MV

Expr2.map

8	7	-4
14	-11	0
-1	MV	-6

Prev mapuniform
Home
Up
Next min

min

Name

min -- Minimum value of multiple expressions

Synopsis

pcrcalc Result = **min**(expression 1, expression 2,... expression n)

expression1-n	ordinal, scalar; expression1-n must have the same data type
	spatial, non spatial
Result	type of expression1-n
	spatial, if all expression1, expression2,...expressionn are non spatial: non spatial

Operation

For each cell, the minimum value of expression1, expression2,...expressionn is determined and assigned to the corresponding cell for Result. As many expressions can be specified as needed.

Notes

A cell with missing value on one or more expressions results in a missing value on the corresponding cell on Result.

Group

This operation belongs to the group of Maximize, minimize

Examples

1.

```
pcrcalc  Result1.map =  min(Expr1.map,Expr2.map)
```

Result1.map

8	6	-4
4	-11	0
-7	MV	MV

Expr1.map

8	6	-2
4	1	0
-7	8	MV

Expr2.map

8	7	-4
14	-11	0
-1	MV	-6

Prev max | Home | Up | Next mod

mod

Name

mod -- Remainder of integer division of values on first expression by values on second expression

Synopsis

pcrcalc Result = expression1 **mod** expression2

expression1	scalar
	spatial, non spatial
expression2	scalar
	spatial, non spatial
Result	scalar
	spatial; non spatial if expression1 and expression2 are non spatial

Operation

For each cell, the value on expression1 is divided (integer division) by the value on expression2. This remainder of this integer division is assigned to the corresponding cell on Result.

Notes

A cell with 0 on expression2 is assigned a missing value on Result. A cell with missing value on expression1 and/or expression2 is assigned a missing value on Result.

Group

This operation belongs to the group of Arithmetic operators

See Also

idiv / or div

Examples

1.

```
pcrcalc  Result.map = Expr1.map mod Expr2.map
```

Result.map

MV	1	-1
0	0.6	0.6
1.2	1.2	0

Expr1.map

3.5	7	-7
0	9.6	9.6
5.2	5.2	56

Expr2.map

0	2	2
7	3	-3
2	-2	7

Prev min | Home | Up | Next ne or !=

ne or !=

Name

ne or != -- Relational-not-equal-to operation

Synopsis

pcrcalc `Result = expression1` **ne** `expression2`

pcrcalc`Result = expression1`**!=**`expression2`

`expression1`	boolean, nominal, ordinal, scalar, directional, ldd spatial, non spatial
`expression2`	type of `expression1` spatial, non spatial
`Result`	boolean spatial; non spatial if `expression1` and `expression2` are non spatial

Operation

For each cell evaluates `expression1` in relation to `expression2`. If the cell value on `expression1` does not equal the value on `expression2` `Result` has a cell value 1 (condition is TRUE) on the corresponding cell; if the cell value on `expression1` equals the value on `expression2` `Result` has a cell value 0 (condition is FALSE).

Notes

A cell with missing value on `expression1` and/or `expression2` is assigned a missing value on `Result`. The **!=** sign is an alternative notation for **ne**.

Group

This operation belongs to the group of Comparison operators

Examples

1.

```
pcrcalc  Result.map = Expr1.map ne Expr2.map
```

Result.map		
1	1	1
1	1	0
1	MV	0

Expr1.map		
4	11	-4
-4	2	3.8
5	MV	0

Expr2.map		
2	-11	-4.1
0	4	3.8
-8	2	0

Prev mod | Home | Up | Next nodirection

nodirection

Name

nodirection -- Expression of directional data type

Synopsis

pcrcalc Result = **nodirection**(expression)

expression	directional
	spatial, non spatial
Result	boolean
	dimension of expression

Operation

The expression cell values represent directions, with values equal to 0 or between 0 and 360 (or 2 *pi* if the global option --radians is set). A cell with no direction (a flat area) has a value -1. Result is a Boolean expression where 1 is TRUE for cells with no direction and 0 is FALSE for cells with a direction.

Each cell without a direction on expression is assigned a 1 (Boolean TRUE) on Result. Cells that have a direction on expression are assigned a 0 (Boolean FALSE) on Result.

Notes

Group

This operation belongs to the group of Missing value creation

See Also

aspect slope

Examples

1.

```
pcrcalc  Result.map = nodirection(Expr.map)
```

Result.map

0	0	0
0	1	0
MV	1	0

Expr.map

280	25	11
68	-1	0
MV	-1	7

Prev ne or != | Home | Up | Next nominal

nominal

Name

nominal -- Data conversion data type nominal data type

Synopsis

pcrcalc Result = **nominal**(expression)

expression	boolean, ordinal, scalar, directional, ldd spatial, non spatial
Result	nominal dimension of expression

Operation

If expression is a PCRaster map or a calculation resulting in a PCRaster map, it is converted: if expression is of one of the data types boolean, ordinal or ldd, the cell values on expression are assigned without change to the corresponding cells on Result; if expression is of data type scalar or direction, the values on expression are truncated.
Or the operation generates a map of nominal data type with one constant value.

If expression is not a PCRaster map, a nominal Result is generated. This is the case if expression is a number. This number must be in the domain of the nominal map type, i.e. a whole value. Result will be a map with the same location attributes as the global clone map; all cells will have the value of expression.

Notes

A cell with missing value on expression is assigned a missing value on Result.

Group

This operation belongs to the group of Conversion and assignment

See Also

Section 2.3.3 Section 2.3.3.3

Examples

1.

```
pcrcalc  Result.map = nominal(Expr.map)
```

Result.map

0	1	3
MV	-3	-2
0	9	8

Expr.map

0	1.5	3.4
MV	-3.2	-2.5
0.01	9.3	8.9

---

normal

Name

normal -- Boolean TRUE cell gets value taken from a normal distribution

Synopsis

```
pcrcalc Result = normal( expression )
```

`expression`	boolean
	spatial, non spatial
`Result`	scalar
	spatial

Operation

A random generator is used to generate the `Result`: for each cell that has a value 1 (TRUE) on `expression`, a value is taken from a normal distribution with mean 0 and standard deviation 1 and assigned to the cell on `Result`. Cells that have a value 0 (FALSE) on `expression` area assigned a missing value.

Notes

Group

This operation belongs to the group of Random number generators; Cells

See Also

Section 6.3.2 Section 6.4.2

Examples

1.

```
pcrcalc  Result.map = normal(uniqueid.map)
```

Result.map

MV	MV	MV	MV	MV
-1.29	0.56	-0.853	-2.38	-0.107
0.518	1.08	0.153	0.15	1.1
MV	-0.996	0.524	1.29	1.14
MV	-1.31	1.38	0.355	2.3

Expr.map

MV	0	MV	MV	0
1	1	1	1	1
1	1	1	1	1
0	1	1	1	1
0	1	1	1	1

Prev nominal | Home | Up | Next not

not

Name

not -- Boolean-NOT operation

Synopsis

pcrcalc Result = **not** expression

`expression`	boolean
	spatial, non spatial
`Result`	boolean
	dimension of `expression`

Operation

The cell values on `expression` are interpreted as Boolean values; where 1 is TRUE and 0 is FALSE. For each cell the Boolean NOT evaluation is performed: if `expression` has a cell value 1 (TRUE) `Result` has a cell value 0 (FALSE) on the corresponding cell; if `expression` has cell value 0 (FALSE) `Result` has cell value 1 (TRUE).

Notes

A cell with missing value on `expression` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Boolean operators

Examples

1.

```
pcrcalc  Result.map = not Expr1.map
```

not

Result.map

0	0	1
1	MV	1
0	0	1

Expr1.map

1	1	0
0	MV	0
1	1	0

Prev Home Next
normal Up or

or

Name

or -- Boolean-OR operation

Synopsis

pcrcalc Result = expression1 **or** expression2

expression1	boolean
	spatial, non spatial
expression2	boolean
	spatial, non spatial
Result	boolean
	spatial; non spatial if expression1 and expression2 are non spatial

Operation

The cell values on expression1 and expression2 are interpreted as Boolean values; where 1 is TRUE and 0 is FALSE. For each cell the Boolean OR evaluation is performed: if expression1 or expression2 or both have a cell value 1 (TRUE) Result has a cell value 1 (TRUE) on the corresponding cell; if expression1 and expression2 have a cell value 0 (FALSE) Result has cell value 0 (FALSE).

Table 1. Cross table of the OR operator.

OR		**expression1**	
		False	True
expression2	False	False	True
	True	True	True

Notes

A cell with missing value on expression1 and/or expression2 is assigned a missing value on Result.

Group

This operation belongs to the group of Boolean operators

Examples

1.

```
pcrcalc  Result.map  = Expr1.map or Expr2.map
```

Result.map

1	1	0
1	MV	MV
1	1	1

Expr1.map

1	1	0
0	MV	0
1	1	0

Expr2.map

0	1	0
1	1	MV
1	1	1

Prev Home Next
not Up order

order

Name

order -- Ordinal numbers to cells in ascending order

Synopsis

pcrcalc Result = **order**(expression)

expression	scalar, ordinal
	spatial
Result	scalar
	spatial

Operation

Let *n* be the number of non missing value cells on expression. These cell values are set in order and numbered on Result in ascending order: the cell with the smallest value on expression is assigned a 1 and the cell with the largest value is assigned a number *n*. Cells on expression with corresponding values are assigned consecutive, unique numbers; the order in which these cells are numbered is arbitrarily chosen.

Notes

A cell with missing value on expression is not considered in the order operation; it is assigned a missing value on Result.

Group

This operation belongs to the group of Data management

Examples

1.

```
pcrcalc  Result.map = order(Expr.map)
```

Result.map

1	10	12	6
2	4	14	5
3	15	11	7
8	9	13	MV

Expr.map

-5	9	9	0
-5	-5	9	0
-5	9	9	2
4	4	9	MV

---

ordinal

Name

ordinal -- Data conversion to the ordinal data type

Synopsis

pcrcalc Result = **ordinal**(expression)

expression	boolean, nominal, scalar, directional, ldd spatial, non spatial
Result	ordinal dimension of expression

Operation

If expression is a PCRaster map or a calculation resulting in a PCRaster map, it is converted: if expression is of one of the data types boolean, nominal or ldd, the cell values on expression are assigned without change to the corresponding cells on Result; if expression is of data type scalar or direction, the values on expression are truncated.
Or the operator generates a map of ordinal data type with one constant value.

If expression has no PCRaster data type, an ordinal Result is generated. This is the case if expression is a number. This number must be in the domain of the ordinal data type, i.e. a whole value. Result will be a map with the same location attributes as the global clone map; all cells will have the value of expression.

Notes

A cell with missing value on expression is assigned a missing value on Result.

Group

This operation belongs to the group of Conversion and assignment

See Also

Section 2.3.3 Section 2.3.3.4

Examples

1.

```
pcrcalc  Result.map = ordinal(Expr.map)
```

Result.map

0	1	3
MV	-3	-2
0	9	8

Expr.map

0	1.5	3.4
MV	-3.2	-2.5
0.01	9.3	8.9

---

path

Name

path -- Path over the local drain direction network downstream to its pit

Synopsis

```
pcrcalc Result = path( ldd, points )
```

`ldd`	ldd
	spatial
`points`	boolean
	spatial
`Result`	boolean
	spatial

Operation

The cell values on `points` are interpreted as Boolean values; where 1 is TRUE and 0 is FALSE. The operation determines for each TRUE cell on `points` the cells which are on the path downstream to its pit. Each path is generated by starting at the TRUE cell on `points` and moving through the consecutively neighbouring downstream cells, following the local drain directions on `ldd`. On `Result`, all cells which are on the path of one or more TRUE cells on `points` are assigned a 1 (TRUE), the cells which are not on a path are assigned a 0 (FALSE).

Notes

A missing value on `points` and/or `ldd` results in a missing value at the corresponding cell on `Result`. A path stops at a missing value cell on `points`. Cells on the downstream path of a missing value on `points` are assigned a 0, unless they are on another path from a TRUE cell on points.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

lddmask

Examples

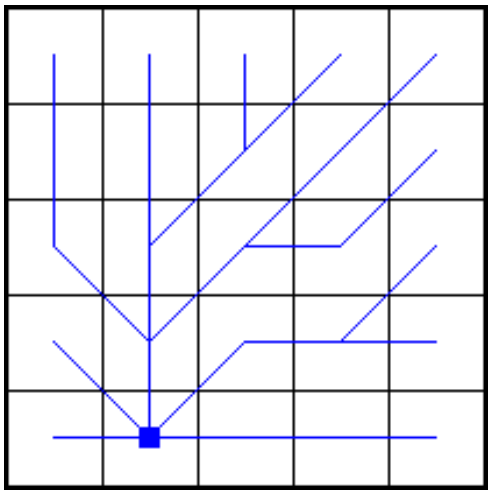
1.

```
pcrcalc  Result.map = path(Ldd.map,Points.map)
```

Result.map

0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
1	1	1	1	0
0	1	1	1	0

Ldd.map



Points.map

0	0	0	0	1
0	0	0	0	0
0	0	0	0	1
1	0	0	0	0
0	0	0	1	0

Prev ordinal | Home | Up | Next pit

pit

Name

pit -- Unique value for each pit cell

Synopsis

pcrcalc Result = **pit**(ldd)

ldd	ldd
	spatial
Result	nominal
	spatial

Operation

A pit is a cell whose neighbours all have a local drain direction in direction of the pit cell. A pit cell doesn't have a local drain direction because all its neighbours are at a higher elevation. Additionally the outflow cell of each catchment on the map, which is a cell at the edge of the map is also a pit. On a local drain direction network pits have a cell value 5.

For every pit cell on the local drain direction network `ldd` an unique number starting with 1 is assigned to the corresponding cell on `Result`; these are cells with a value 5 on `ldd`. The other cells on `ldd` do have a local drain direction and are assigned a 0 value on `Result`.

Notes

A missing value on `ldd` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

lddmask

Examples

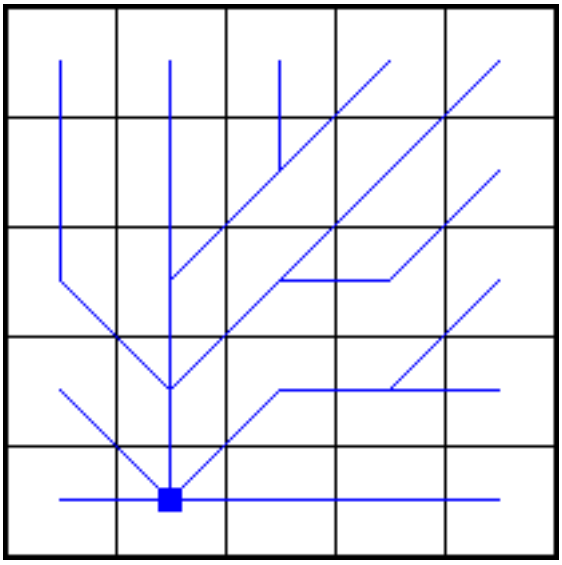
1.

```
pcrcalc  Result.map = pit(Ldd.map)
```

Result.map

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	1	0	0	0

Ldd.map



Prev path | Home Up | Next plancurv

plancurv

Name

plancurv -- Planform curvature calculation using a DEM

Synopsis

pcrcalc [option] Result = **plancurv**(dem)

dem	scalar
	spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell (see also notes)

--unittrue

horizontal and vertical scale is measured in true distance; values on dem are interpreted as real heights (default).

--unitcell

horizontal and vertical scale is measured in number of cell lengths; values on dem are interpreted as number of cell lengths.

Operation

Calculates the planform curvature (i.e. the curvature transverse to the slope) on the elevation map dem. For each cell, the curvature is calculated in a 3 x 3 cell window, where the cell under consideration is in the center of the window.

The planform curvature on Result is the change in slope per distance in horizontal direction, in direction of the slope. It is negative at concave slopes and positive at convex slopes. The slope is d*z*/d*x*, which is the increase in height (vertical direction d*z*) per distance in horizontal direction (d*x*). The planform curvature is the change in slope per distance in horizontal direction, so it is d*z*/d*x* ^{2}. For a correct calculation of the planform curvature, the scale for the horizontal distance on dem and the

vertical distance (height) on `dem` must be the same and the global option --unittrue must be set (it is default). In that case, the scales of *z* and *x* correspond and the planform curvature on `Result` is in 1/d*x*.

The planform curvature is calculated according to the equation given by [zevenbergen87].

Notes

If a cell has a missing value on `dem`, a missing value is assigned to `Result`, in any case.

For each cell, the planform curvature is calculated using its 8 neighbours in a 3 x 3 cells window. Elevation in all these cells must be known, else the planform curvature calculation can not be performed. It may occur that one of these values is unknown: this is the case if a surrounding cell is a missing value or if the centre cell is at the edge of the map resulting in the absence of some surrounding cells. If this occurs the **plancurv** operator uses a built in neighbourhood interpolator to fill these missing values or absent cells in; this will make calculation of the planform curvature for the centre cell still possible. For each missing value cell or absent cell, the elevation is determined by taking the average elevation of non missing value cells in a 3 x 3 cell window, with the missing value cell or absent cell in the centre of the window.

Group

This operation belongs to the group of Derivatives of elevation maps

See Also

profcurv

Examples

1.
```
pcrcalc  Result.map = plancurv(Dem.map)
```
```
 Result.map                Dem.map
```

0	0	-0	MV	-0.01
0	0	-0	MV	MV
0.01	0.02	0	0	-0.05
0.03	-0.02	0.04	0.06	0.02
-0.1	-0.07	0.09	-0.03	-0.06

70	70	80	MV	120
70	70	90	MV	MV
70	70	100	140	280
180	160	110	160	320
510	440	300	400	480

Prev pit | Home Up | Next pred

pred

Name

pred -- Ordinal number of the next lower ordinal class

Synopsis

pcrcalc Result = **pred**(expression)

expression	ordinal
	spatial
Result	ordinal
	spatial

Operation

The result of the operation depends on wheter expression has a legend or not. If expression has a legend, the legend determines the domain of expression: the domain consists of the ordinal numbers linked to the classes in the legend. This domain with these ordinal classes is also assigned to Result. Cells on Result may have values in this domain. For each expression cell value the first lower ordinal number which is in the domain is determined. This is assigned to the corresponding cell on Result.

If expression does not have a legend an ordinal number is assigned to Result which is the ordinal number on expression minus 1, on a cell-by-cell basis.

Notes

A cell on expression with missing value is assigned a missing value on Result.

Group

This operation belongs to the group of Data management

Examples

1.

```
pcrcalc  Result1.map = pred(Expr.map)
```

Result1.map

-6	8	8	-1
-6	-6	8	-1
-6	8	8	1
3	3	8	MV

Expr.map

-5	9	9	0
-5	-5	9	0
-5	9	9	2
4	4	9	MV

Prev plancurv | Home Up | Next profcurv

profcurv

Name

profcurv -- Profile curvature calculation using a DEM

Synopsis

pcrcalc [option] Result = **profcurv**(dem)

dem	scalar
	spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell (see also notes)

--unittrue

horizontal and vertical scale is measured in true distance; values on dem are interpreted as real heights (default).

--unitcell

horizontal and vertical scale is measured in number of cell lengths; values on dem are interpreted as number of cell lengths.

Operation

Calculates the profile curvature (i.e. the curvature in the direction of the slope) on the elevation map dem. For each cell, the curvature is calculated in a 3 x 3 cell window, where the cell under consideration is in the center of the window.

The profile curvature on Result is the change in slope per distance in horizontal direction, in direction of the slope. It is negative at concave slopes and positive at convex slopes. The slope is d*z*/d*x*, which is the increase in height (vertical direction d*z*) per distance in horizontal direction (d*x*). The profile curvature is the change in slope per distance in horizontal direction, so it is d*z*/d*x* ^{2}. For a correct calculation of the profile curvature, the scale for the horizontal distance on dem and the vertical distance

(height) on `dem` must be the same and the global option --unittrue must be set (it is default). In that case, the scales of *z* and *x* correspond and the profile curvature on `Result` is in 1/d*x*.

The profile curvature is calculated according to the equation given by [zevenbergen87].

Notes

If a cell has a missing value on `dem`, a missing value is assigned to `Result`, in any case.

For each cell, the profile curvature is calculated using its 8 neighbours in a 3 x 3 cells window. Elevation in all these cells must be known, else the profile curvature calculation can not be performed. It may occur that one of these values is unknown: this is the case if a surrounding cell is a missing value or if the centre cell is at the edge of the map resulting in the absence of some surrounding cells. If this occurs the **profcurv** operator uses a built in neighbourhood interpolator to fill these missing values or absent cells in; this will make calculation of the profile curvature for the centre cell still possible. For each missing value cell or absent cell, the elevation is determined by taking the average elevation of non missing value cells in a 3 x 3 cell window, with the missing value cell or absent cell in the centre of the window.

Group

This operation belongs to the group of Derivatives of elevation maps

See Also

plancurv

Examples

1.

```
pcrcalc  Result.map = profcurv(Dem.map)
```

`Result.map`

0	-0	-0.01	MV	-0.02
0	-0.01	-0.01	MV	MV
-0.05	-0.03	-0	-0.04	0.02
-0.08	-0.08	-0.07	-0.07	0.02
0.14	0.08	0.04	0.08	0.08

`Dem.map`

70	70	80	MV	120
70	70	90	MV	MV
70	70	100	140	280
180	160	110	160	320
510	440	300	400	480

Prev pred | Home | Up | Next rounddown

rounddown

Name

rounddown -- Rounding down of cellvalues to whole numbers

Synopsis

pcrcalc Result = **rounddown**(expression)

expression	scalar
	spatial, non spatial
Result	scalar
	dimension of expression

Operation

For each cell, the value on expression is rounded downwards: the next whole value that is less than or equal to the value on expression is assigned to Result.

Notes

Input values can be positive or negative.

A cell with a missing value on expression is assigned a missing value on Result.

Group

This operation belongs to the group of Rounding

Examples

1.
```
pcrcalc  Result.map = rounddown(Expr.map)
```

Result.map

0	1	3
MV	-4	-3
0	9	8

Expr.map

0	1.5	3.4
MV	-3.2	-2.5
0.01	9.3	8.9

Prev profcurv | Home | Up | Next roundoff

roundoff

Name

roundoff -- Rounding off of cellvalues to whole numbers

Synopsis

pcrcalc Result = **roundoff**(expression)

expression	scalar
	spatial, non spatial
Result	scalar
	dimension of expression

Operation

For each cell, the value on expression is rounded off: the whole value whose difference with the value on expression is smallest is assigned to Result. If a value on expression is exactly halfway between two whole numbers, the number with the greatest absolute magnitude is assigned to Result.

Notes

Input values can be positive or negative. A cell with missing value on expression is assigned a missing value on Result.

Group

This operation belongs to the group of Rounding

Examples

1.

```
pcrcalc  Result.map = roundoff(Expr.map)
```

Result.map

0	2	3
MV	-3	-3
0	9	9

Expr.map

0	1.5	3.4
MV	-3.2	-2.5
0.01	9.3	8.9

Prev rounddown | Home | Up | Next roundup

roundup

Name

roundup -- Rounding up of cellvalues to whole numbers

Synopsis

pcrcalc Result = **roundup**(expression)

expression	scalar
	spatial, non spatial
Result	scalar
	spatial, non spatial

Operation

For each cell, the `expression` value is rounded upwards: the next whole value that is greater than or equal to the value on `expression` is assigned to `Result`.

Notes

Input values can be positive or negative. A cell with missing value on `expression` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Rounding

Examples

1.
```
pcrcalc  Result.map = roundup(Expr.map)
```

Result.map

0	2	4
MV	-3	-2
1	10	9

Expr.map

0	1.5	3.4
MV	-3.2	-2.5
0.01	9.3	8.9

Prev roundoff | Home | Up | Next scalar

scalar

Name

scalar -- Data conversion to the scalar data type

Synopsis

pcrcalc Result = **scalar**(expression)

expression	boolean, nominal, ordinal, directional, ldd spatial, non spatial
Result	scalar dimension of expression

Operation

If expression is a PCRaster map or a calculation resulting in a PCRaster map, it is converted: the cell values of expression are assigned without change to the corresponding cells on Result. Or it generates a map of scalar data type with one constant value.

If expression has no PCRaster data type, a Result with data type scalar is generated. This is the case if expression is a number or a calculation with numbers. Result will be a map with the same location attributes as the global clone map; all cells will have the value of expression.

Notes

A cell with missing value on expression is assigned a missing value on Result.

Group

This operation belongs to the group of Conversion and assignment

See Also

Section 2.3.3 Section 2.3.3.5

Examples

1.

```
pcrcalc  --clone Expr1.map Result.map = scalar(1)
```

Result.map

1	1	1
1	1	1
1	1	1

Expr1.map

1	1	0
0	MV	0
1	1	0

Prev roundup | Home Up | Next sin

sin

Name

sin -- Sine

Synopsis

pcrcalc [option] Result = **sin**(expression)

expression	directional, scalar
	spatial, non spatial
Result	scalar
	dimension of expression

Options

if expression is a number: --degrees or --radians

--degrees

direction is measured in degrees (default)

--radians

direction is measured in radians

Operation

For each cell, calculates the sine of the cell value on expression and assigns it to Result.

Notes

A cell with missing value on expression is assigned a missing value on Result.

If expression is of directional data type, a cell on expression without a direction (cell value -1) is assigned a missing value.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.

```
pcrcalc  Result.map = sin(Expr.map)
```

Result.map

0.5	0.342	0.311
MV	0	-0.982
0.707	0.144	0.0175

Expr.map

30	20	18.1
MV	0	281
45	8.3	1

Prev scalar | Home | Up | Next slope

slope

Name

slope -- Slope of cells using a digital elevation model

Synopsis

pcrcalc [option] Result = **slope**(dem)

dem	scalar
	spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell (see also notes)

--unittrue

horizontal and vertical scale is measured in true distance; values on dem are interpreted as real heights (default).

--unitcell

horizontal and vertical scale is measured in number of cell lengths; values on dem are interpreted as number of cell lengths.

Operation

For each cell, calculates the slope on basis of the elevation dem of its eight nearest neighbours in a 3 x 3 cell window. The third-order finite difference method is used, proposed by Horn (1981), also used by [skidmore89]. The slope on Result is given in dz/d*x*, which is the increase in height (vertical direction dz) per distance in horizontal direction (d*x*), yielding a value between 0 and 1. This result value is often reffered to as a percentage. Thus if slope returns a value of 0.12, one says a slope value of 12 %.

Notes

Always set the option --unittrue; the option --unitcell is only used in very very special cases. In addition, note that for a correct calculation of the slope the scales for the horizontal distance on your map and the vertical distance (height) on `dem` must be the same. For instance, choose for both distances metres.

If a cell has a missing value on `dem`, a missing value is assigned to `Result`, in any case.

For each cell, the slope is calculated using its 8 neighbours in a 3 x 3 cells window. Elevation in all these cells must be known, else the finite difference method can not be performed. It may occur that one of these values is unknown: this is the case if a surrounding cell is a missing value or if the centre cell is at the edge of the map resulting in the absence of some surrounding cells. If this occurs the **slope** operator uses a built in neighbourhood interpolator to fill these missing values or absent cells in; this will make calculation of the slope for the centre cell still possible. For each missing value cell or absent cell, the elevation is determined by taking the average elevation of non missing value cells in a 3 x 3 cell window, with the missing value cell or absent cell in the centre of the window.

Group

This operation belongs to the group of Derivatives of elevation maps

See Also

nodirection aspect

Examples

1.

```
pcrcalc  Result.map = slope(Dem.map)
```

Result.map

.0118	0.114	0.394	MV	0.673
0.13	0.206	0.604	MV	MV
1.3	0.775	0.643	1.73	1.87
3.73	3.54	2.58	3.02	2.36
2.76	3.07	2.59	2.66	1.65

Dem.map

70	70	80	MV	120
70	70	90	MV	MV
70	70	100	140	280
180	160	110	160	320
510	440	300	400	480

Prev Home Next
sin Up slopelength

slopelength

Name

slopelength -- Accumulative-friction-distance of the longest accumulative-friction-path upstream over the local drain direction network cells against waterbasin divides

Synopsis

pcrcalc [option] Result = **slopelength**(ldd, friction)

ldd	ldd
	spatial
friction	scalar
	spatial, non spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell:

--unittrue

distance is measured in true distance (default)

--unitcell

distance is measured in number of cell lengths

Operation

For a cell on a local drain direction network its catchment is made up of the cell itself (the outflow cell) and all cells that drain to the cell. The catchment is circumscribed by the divide. Call the cells in the catchment against this divide the divide cells of the catchment: neighbouring cells of the divide cells are cells downstream of the divide cell in the same catchment as the divide cell or cells on the other side of the divide. As a result there are no cells that drain to a divide cell. For each divide cell a downstream path can be defined which begins at the centre of the divide cell, follows the local drain directions in downstream direction and stops at the centre of the outflow cell of the catchment.

For all cells the following procedure is performed: using the local drain direction network on `ldd` the divide cells of the cell its catchment are determined, where the cell itself is the outflow cell of the catchment. Now, for each divide cell the accumalative-friction distance over its downstream path to the outflow cell is calculated as follows: an amount of friction moves through the consecutively neighbouring downstream cells, following the downstream path of the divide cell, until it reaches the centre of the outflow cell under consideration. It accumulates each time it travels from one cell to its downstream next starting with an amount of 0 at the divide cell. The amount of friction which accumulates per unit distance when moving from one cell to the next is specified by the cell values on `friction`. Using the values on this expression, accumulation of friction when travelling from one cell to its first downstream cell is calculated as follows:

Let `friction`(sourcecell) and `friction`(destinationcell) be the `friction` values at the cell where friction is transported from and at its downstream cell where friction is transported to, respectively. While moving from the source cell to the destination cell the amount of accumulated friction is incremented with:

distance x {(`friction`(sourcecell)+`friction`(destinationcell)}/2

where distance is the distance between the sourcecell and the destination cell. This distance equals the cell length if the source cell and the destination cell are neighbours in horizontal or vertical directions; it equals sqrt(2) multiplied with the cell length if the cells are neighbours in diagonal directions.

For all divide cells the accumulated-friction-distance of the downstream path to the outflow cell is determined. The accumulated-friction-distance of the downstream path resulting in the greatest accumulated-friction- distance is assigned to the outflow cell on `Result`. This procedure is performed for each cell, where each cell is regarded as a outflow cell of an catchment with one or several divide cells.

Notes

The values on `friction` must be larger than zero. A cell with a missing value on `ldd` is assigned a missing value on `Result`. A cell with missing value on `friction` is assigned a missing value on `Result`; all cells which are on the downstream path of the missing value are also assigned a missing value on `Result`, unless they also make part of another downstream path.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

Section 4.4.4 lddist Section 6.2.3 lddmask

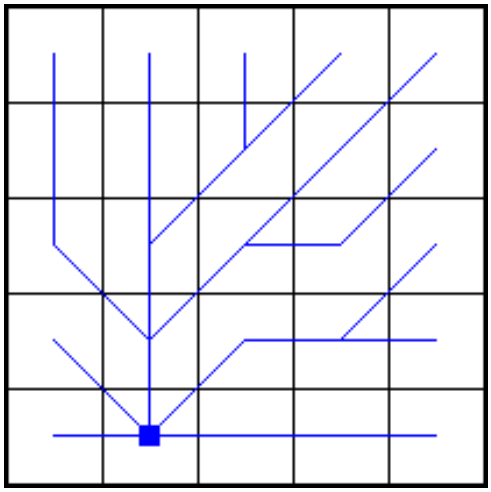
Examples

1.
```
pcrcalc  Result1.map = slopelength(Ldd2.map,1)
```

`Result1.map`

0	0	0	0	0
2	2	2.83	2.83	0
4	5.66	5.66	2.83	0
0	8.49	4.83	2.83	0
0	10.5	4	2	0

`Ldd2.map`

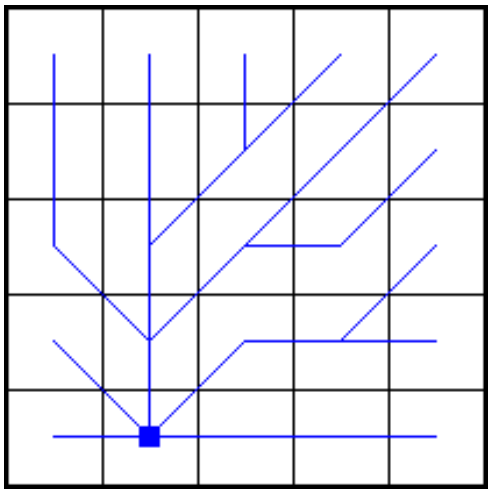


2.
```
pcrcalc  Result2.map = slopelength(Ldd2.map,FrictMat.map)
```

Result2.map

0	0	0	0	0
400	2	11.3	11.3	0
601	18.4	22.6	11.3	0
0	604	4.83	2.83	0
0	606	MV	MV	0

Ldd2.map



FrictMat.map

200	1	4	4	4
200	1	4	4	4
1	1	4	4	1
8	1	1	1	1
2	1	5	MV	200

Prev slope | Home Up | Next spread

spread

Name

spread -- Total friction of the shortest accumulated friction path over a map with friction values from source cell to cell under consideration

Synopsis

pcrcalc [option] Result = **spread**(points, initialfrictiondist, friction)

points	boolean, nominal, ordinal
	spatial
initialfrictiondist	scalar
	spatial, non spatial
friction	scalar
	spatial, non spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell

--unittrue

distance is measured in true distance (default)

--unitcell

distance is measured in number of cell lengths

Operation

The expression points identifies those cells from which the shortest accumulated friction path to every cell centre is calculated. The spreading for determination of these friction-distances starts at the centre of cells which have a non zero value on points. The initial friction-distance (at the start of the spreading) is taken from the values at these point cells on initialfrictiondist. During spreading a path is followed over the consecutive neighbouring

cells. While following this path the friction-distance increases. The increase of friction-distance per unit distance is specified by the cell values on `friction`. Using these values, increase when travelling from one cell to its neighbouring cell is calculated as follows: Let `friction`(sourcecell) and `friction`(destinationcell) be the `friction` values at the cell where is moved from and where is moved to, respectively. While moving from the source cell to the destination cell the increase of friction- distance is:

distance x {(`friction`(sourcecell)+`friction`(destinationcell)}/2

where distance is the distance between the sourcecell and the destination cell. This distance equals the cell length if the source cell and the destination cell are neighbours in horizontal or vertical directions; it equals sqrt(2) multiplied by the cell length if the cells are neighbours in diagonal directions.

During operation of the command, the spreading is executed from all non zero cells on `points`, over all possible paths. For determination of the accumulated friction-distance cell values on `Result`, for each cell the path from a non zero cell on `points` is chosen with the shortest friction-distance. So during the execution of the **spread** operation, for each cell, the friction-distance for each possible path from the non zero cells on `points` to the cell under consideration is calculated and then the path with the shortest friction-distance is chosen. On `Result` each cell has a value which is the friction-distance covered when moving over this shortest path from a non zero cell on `points`.

Notes

The values on `friction` must be larger than zero.

Missing value cells on `points`, `initialfrictiondist` and `friction` are assigned a missing value on `Result`. Potential shortest paths that cross missing value cells on `points`, `initialfrictiondist` or `friction` are ignored.

If no path is found for a cell (for instance if the cell is surrounded by missing values), a missing value is assigned to that cell on `Result`.

Group

This operation belongs to the group of Neighbourhood operators; spread operators

See Also

Section 4.4.4 lddist slopelength

Examples

1.
```
pcrcalc  Result1.map = spread(Points.map,0,1)
```

2.

Result1.map

2.83	2	2	2	0
2	0	0	0	MV
2	0	2	2	2.83
2	0	2	4	4.83
2	0	2	4	6

Points.map

0	0	0	0	6
0	1	1	2	MV
0	4	0	0	0
0	2	0	0	0
0	3	0	0	0

```
pcrcalc  Result2.map = spread(Points2.map,Initial2.map,FrictMat2.map)
```

Result2.map

6.83	205	6.83	6	4
204	4	4.83	9.83	9
2.83	2	7	11.9	17
2	0	4	207	221
2.83	MV	5.66	209	413

Points2.map

0	0	0	0	2
0	0	0	0	0
0	0	0	0	0
0	1	0	0	0
0	0	0	0	0

Initial2.map

8	8	8	8	4
8	8	8	8	8
8	8	8	8	8
0	0	8	8	8
0	0	8	8	8

FrictMat2.map

1	200	1	1	1
200	1	1	4	4
1	1	4	4	4
1	1	3	200	200
1	MV	3	200	4

Prev
slopelength

Home
Up

Next
spreadldd

spreadldd

Name

spreadldd -- Total friction of the shortest accumulated friction downstream path over map with friction values from an source cell to cell under consideration

Synopsis

pcrcalc [option] Result = **spreadldd**(ldd, points, initialfrictiondist, friction)

ldd	ldd
	spatial
points	boolean, nominal, ordinal
	spatial
initialfrictiondist	scalar
	spatial, non spatial
friction	scalar
	spatial, non spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell

--unittrue

distance is measured in true distance (default)

--unitcell

distance is measured in number of cell lengths

Operation

The expression points identifies those cells from which the shortest friction-distance to every cell centre is calculated. The spreading for determination of these friction-distances starts at the centre of cells which have a non zero value on points. The initial friction distance (at the start of the spreading) is taken from the values at these point cells on initialfrictiondist. During spreading a path is followed over the consecutive neighbouring cells. While following this path the friction-distance increases. The increase of friction-distance per unit distance is specified by the cell values on friction. Using these values, increase when travelling from one cell to its neighbouring cell is calculated as follows: Let friction(sourcecell) and friction(destinationcell) be the friction values at the cell where is moved from and where is moved to, respectively. While moving from the source cell to the destination cell the increase of friction- distance is:

distance x {(friction(sourcecell)+friction(destinationcell)}/2

where distance is the distance between the sourcecell and the destination cell. This distance equals the cell length if the source cell and the destination cell are neighbours in horizontal or vertical directions; it equals sqrt(2) multiplied by the cell length if the cells are neighbours in diagonal directions.

During operation of the command, the spreading is executed from all non zero cells on points, over all possible paths. For determination of the friction-distance cell values on Result, for each

cell the path from a non zero cell on `points` is chosen with the shortest friction-distance. So during the execution of the **spreadldd** operation, for each cell, the friction-distance for each possible path from the non zero cells on `points` to the cell under consideration is calculated and then the path with the shortest friction-distance is chosen. On `Result` each cell has a value which is the friction-distance covered when moving over this shortest path from a non zero cell on `points`.

Notes

The values on `friction` must be larger than zero.

Missing value cells on `points`, `initialfrictiondist` and `friction` are assigned a missing value on `Result`. Additionally, potential shortest paths that cross missing value cells are ignored.

If a cell has no source cell (i.e. a non zero cell value on `points`) on its upstream path or paths it is assigned a missing value.

Group

This operation belongs to the group of Neighbourhood operators; spread operators

See Also

Section 4.4.4 lddist slopelength

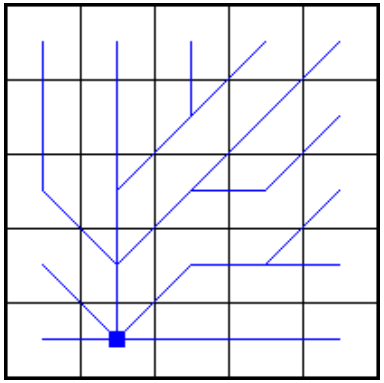
Examples

1.
```
pcrcalc  Result1.map = spreadldd(Ldd2.map, Points1.map,0,1)
```

Result1.map

0	0	0	MV	MV
2	2	2	0	0
4	4	2.83	2.83	0
0	5.66	4	2	0
MV	2.83	MV	MV	MV

Ldd2.map



Points1.map

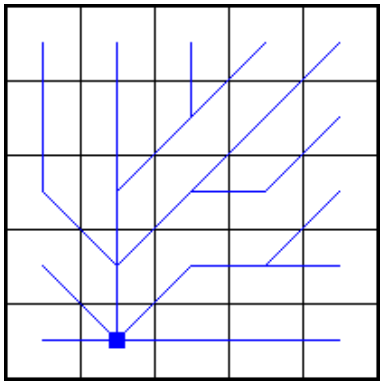
1	2	3	0	MV
0	0	0	4	5
0	0	0	0	6
8	0	0	0	7
0	0	0	0	0

2.
```
pcrcalc  Result2.map = spreadldd(Ldd2.map, Points2.map,Initial.map,FrictMat.map)
```

Result2.map

MV	0	MV	0	0
MV	0	2.83	MV	MV
MV	4	MV	MV	0
250	8	4.83	2.83	MV
MV	9.07	400	200	0

Ldd2.map



Points2.map

0	2	0	6	4
0	5	0	0	0
0	0	0	0	3
1	0	0	0	0
0	0	0	0	7

Initial.map

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
250	0	0	0	0
0	0	0	0	0

FrictMat.map

100	2	1	1	1
100	2	1	MV	1
100	2	1	1	1
2	2	1	1	1
2	2	100	100	100

Prev
spread

Home
Up

Next
spreadlddzone

spreadlddzone

Name

spreadlddzone -- Shortest friction-distance path over map with friction from a source cell to cell under consideration, only paths in downstream direction from the source cell are considered

Synopsis

pcrcalc [option] Result = **spreadlddzone**(ldd, points, initialfrictiondist, friction)

ldd	ldd
	spatial
points	nominal, ordinal, boolean
	spatial
initialfrictiondist	scalar
	spatial, non spatial
friction	scalar
	spatial, non spatial
Result	points
	spatial

Options

--unittrue or --unitcell

--unittrue

distance is measured in true distance (default)

--unitcell

distance is measured in number of cell lengths

Operation

The expression points identifies those cells from which the shortest friction-distance to every cell centre is calculated. The spreading for determination of these friction-distances starts at the centre of cells which have a non zero value on points. The initial friction distance (at the start of the spreading) is taken from the values at these point cells on initialfrictiondist. During spreading a path is followed over the consecutive neighbouring cells. While following this path the friction-distance increases. The increase of friction-distance per unit distance is specified by the cell values on friction. Using these values, increase when travelling from one cell to its neighbouring cell is calculated as follows: Let friction(sourcecell) and friction(destinationcell) be the friction values at the cell where is moved from and where is moved to, respectively. While moving from the source cell to the destination cell the increase of friction- distance is:

distance x {(friction(sourcecell)+friction(destinationcell)}/2

where distance is the distance between the sourcecell and the destination cell. This distance equals the cell length if the source cell and the destination cell are neighbours in horizontal or vertical directions; it equals sqrt(2) multiplied by the cell length if the cells are neighbours in diagonal directions.

During operation of the command, the spreading is executed from all non zero cells on points, over all possible paths. For determination of the friction-distance cell values on Result, for each

cell the path from a non zero cell on `points` is chosen with the shortest friction-distance. So during the execution of the **spreadlddzone** operation, for each cell, the friction-distance for each possible path from the non zero cells on `points` to the cell under consideration is calculated and then the path with the shortest friction-distance is chosen. On `Result` each cell is assigned the `points` cell value of the cell where the shortest path to the cell begins.

Notes

The values on `friction` must be larger than zero.

Missing value cells on `points`, `initialfrictiondist` and `friction` are assigned a missing value on `Result`. Additionally potential shortest paths that cross missing value cells on `points`, `initialfrictiondist` or `friction` are ignored.

Group

This operation belongs to the group of Neighbourhood operators; spread operators

See Also

Section 4.4.4 lddist slopelength

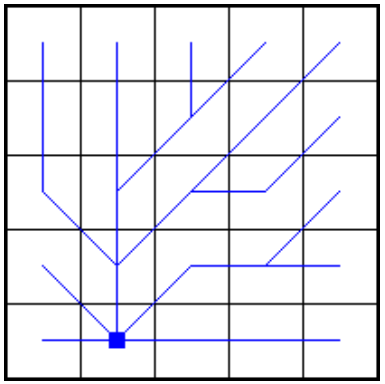
Examples

1.
```
pcrcalc  Result1.map = spreadlddzone(Ldd2.map, Points1.map,0,1)
```

`Result1.map`

1	2	3	0	MV
1	2	3	4	5
1	2	4	5	6
8	4	7	7	7
0	8	0	0	0

`Ldd2.map`



`Points1.map`

1	2	3	0	MV
0	0	0	4	5
0	0	0	0	6
8	0	0	0	7
0	0	0	0	0

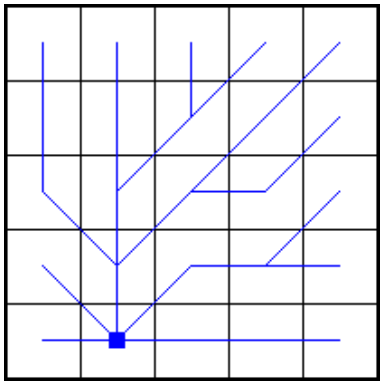
2.
```
pcrcalc  Result2.map = spreadlddzone(Ldd2.map, Points2.map,Initial.map,FrictMat.map)
```

spreadlddzone

Result2.map

0	2	0	6	4
0	5	6	MV	0
0	5	0	0	3
1	5	3	3	0
0	3	7	7	7

Ldd2.map



Points2.map

0	2	0	6	4
0	5	0	0	0
0	0	0	0	3
1	0	0	0	0
0	0	0	0	7

Initial.map

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
250	0	0	0	0
0	0	0	0	0

FrictMat.map

100	2	1	1	1
100	2	1	MV	1
100	2	1	1	1
2	2	1	1	1
2	2	100	100	100

Prev
spreadldd

Home
Up

Next
spreadmax

spreadmax

Name

spreadmax -- Total friction of the shortest accumulated friction path over a map with friction values from a source cell to cell under consideration

Synopsis

pcrcalc Result = **spreadmax**(expression)

expression	nominal, ordinal, scalar, directional, ldd
	spatial, non spatial
Result	boolean
	dimension of expression

Operation

Identical to spread and spreadzone but with a fourth parameter, a maximum spread distance. Areas that are not reached are given the value for MV for spreadmax and 0 for spreadmaxzone.

spreadzone

Name

spreadzone -- Shortest friction-distance path over a map with friction from an identified source cell or cells to the cell under consideration

Synopsis

pcrcalc [option] Result = **spreadzone**(points, initialfrictiondist, friction)

points	boolean, nominal, ordinal
	spatial
initialfrictiondist	scalar
	spatial, non spatial
friction	scalar
	spatial, non spatial
Result	points
	spatial

Options

--unittrue or --unitcell

--unittrue

distance is measured in true distance (default)

--unitcell

distance is measured in cells

Operation

The expression points identifies those cells from which the shortest friction-distance to every cell centre is calculated. The spreading for determination of these friction-distances starts at the centre of cells which have a non zero value on points. The initial friction distance (at the start of the spreading) is taken from the values at these point cells on initialfrictiondist. During spreading a path is followed over the consecutive neighbouring cells. While

following this path the friction-distance increases. The increase of friction-distance per unit distance is specified by the cell values on `friction`. Using these values, increase when travelling from one cell to its neighbouring cell is calculated as follows: Let `friction`(sourcecell) and `friction`(destinationcell) be the `friction` values at the cell where is moved from and where is moved to, respectively. While moving from the source cell to the destination cell the increase of friction- distance is:

distance x {`friction`(sourcecell)+`friction`(destinationcell)}/2

where distance is the distance between the sourcecell and the destination cell. This distance equals the cell length if the source cell and the destination cell are neighbours in horizontal or vertical directions; it equals sqrt(2) multiplied by the cell length if the cells are neighbours in diagonal directions.

During operation of the command, the spreading is executed from all non zero cells on `points`, over all possible paths. For determination of the friction-distance cell values on `Result`, for each cell the path from a non zero cell on `points` is chosen with the shortest friction-distance. So during the execution of the **spread** operation, for each cell, the friction-distance for each possible path from the non zero cells on `points` to the cell under consideration is calculated and then the path with the shortest friction-distance is chosen. On `Result` each cell is assigned the `points` cell value of the cell where the shortest path to the cell begins. Cells for which no path is found are assigned a value 0.

Notes

The values on `friction` must be larger than zero.

Missing value cells on `points`, `initialfrictiondist` and `friction` are assigned a missing value on `Result`. Potential shortest paths that cross missing value cells on `points`, `initialfrictiondist` or `friction` are are not considered.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

Section 4.4.4 lddist slopelength

Examples

1.
```
pcrcalc  Result1.map = spreadzone(Points.map,0,1)
```

2.

Result1.map

1	1	1	2	6
1	1	1	2	MV
4	4	4	2	2
2	2	2	2	2
3	3	3	3	3

Points.map

0	0	0	0	6
0	1	1	2	MV
0	4	0	0	0
0	2	0	0	0
0	3	0	0	0

```
pcrcalc  Result2.map=spreadzone(Points2.map,Initial2.map,FrictMat2.map)
```

Result2.map

1	1	1	2	2
1	1	1	1	2
1	1	1	1	2
1	1	1	1	2
1	MV	1	1	1

Points2.map

0	0	0	0	2
0	0	0	0	0
0	0	0	0	0
0	1	0	0	0
0	0	0	0	0

Initial2.map

8	8	8	8	4
8	8	8	8	8
8	8	8	8	8
0	0	8	8	8
0	0	8	8	8

FrictMat2.map

1	200	1	1	1
200	1	1	4	4
1	1	4	4	4
1	1	3	200	200
1	MV	3	200	4

Prev spreadmax
Home Up
Next sqr

sqr

Name

sqr -- Square

Synopsis

pcrcalc Result = **sqr**(expression)

expression	scalar
	spatial, non spatial
Result	scalar
	dimension of expression

Operation

For each cell, calculates the square of the expression cell value and assigns it to Result.

Notes

A cell with a missing value on expression is assigned a missing value on Result.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.

```
pcrcalc  Result.map = sqr(Expr.map)
```

Result.map

MV	4	0.25
4	0	9.61
65.6	64	0.25

Expr.map

MV	2	0.5
-2	0	3.1
8.1	8	-0.5

Prev spreadzone | Home | Up | Next sqrt

sqrt

Name

sqrt -- Square root

Synopsis

```
pcrcalc Result = sqrt( expression )
```

`expression`	scalar
	spatial, non spatial
`Result`	scalar
	dimension of `expression`

Operation

For each cell, calculates the square root of the `expression` cell value and assigns it to `Result`.

Notes

The cell values on `expression` must be equal to or greater than 0. Negative values are assigned a missing value on `Result`. A cell with a missing value on `expression` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.
```
pcrcalc  Result.map = sqrt(Expr.map)
```

Result.map

0	4	3
MV	0.1	MV
4.12	0.837	2.24

Expr.map

0	16	9
-81	0.01	MV
17	0.7	5

Prev sqr | Home | Up | Next streamorder

streamorder

Name

streamorder -- Stream order index of all cells on a local drain direction network

Synopsis

```
pcrcalc Result = streamorder( ldd )
```

`ldd`	ldd
	spatial
`Result`	ordinal
	spatial

Operation

The classification of stream networks was originally developed by Horton, and modified by Strahler [strahler64]. Following the scheme of Strahler, **streamorder** designates an order 1 to the smallest channels, which are the cells with no upstream cells connected to that cell. Where two channels of order 1 join, a channel of order 2 results downstream. In general, where two channels of order i join, a channel of order i+1 results.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

accuflux

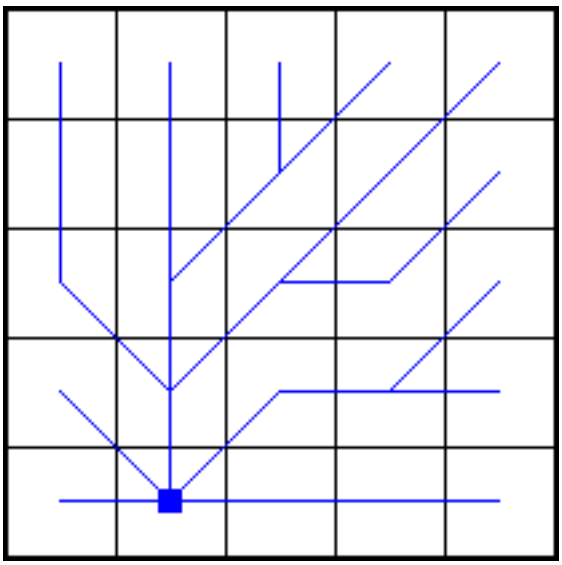
Examples

1.
```
pcrcalc  Result.map = streamorder(Ldd.map)
```

Result.map

1	1	1	1	1
1	1	2	1	1
1	2	2	1	1
1	3	2	2	1
1	3	1	1	1

Ldd.map



Prev sqrt | Home Up | Next subcatchment

subcatchment

Name

subcatchment -- (Sub-)Catchment(s) (watershed, basin) of each one or more specified cells

Synopsis

pcrcalc Result = **subcatchment**(ldd, points)

ldd	ldd spatial
points	boolean, nominal, ordinal spatial
Result	type of points spatial

Operation

The local drain direction for each cell is defined by ldd. For each non zero value on points its catchment is determined and all cells in its catchment are assigned this non zero value. This procedure is performed for all cells with a non zero value on points, but there is one important exception: subcatchments are also identified: if the catchment of a non zero cell on points contains another non zero cell value on points, the (smaller) catchment of the latter non zero cell is identified instead of the (larger) enclosing catchment.

The operation is performed as follows: for each cell its downstream path is determined which consists of the consecutively neighbouring downstream cells on ldd. On Result each cell is assigned the non zero points cell value which is on its path and which is nearest downstream. If all cells on the downstream path of a cell have a value 0 on points a 0 is assigned to the cell on Result.

Notes

A cell with missing value on ldd is assigned a missing value on Result.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

See Also

catchment lddmask

Examples

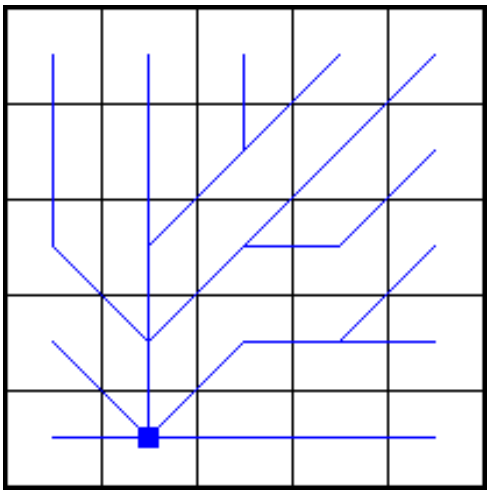
1.

```
pcrcalc  Result.map = subcatchment(Ldd.map,Points.map)
```

Result.map

3	4	4	4	3
3	4	4	3	3
3	4	3	3	2
0	3	2	2	2
0	0	0	0	1

Ldd.map



Points.map

0	0	0	0	0
0	0	0	MV	0
0	4	0	0	0
0	3	2	0	0
0	0	0	0	1

Prev streamorder | Home Up | Next succ

succ

Name

succ -- Ordinal number of the next higher ordinal class

Synopsis

```
pcrcalc Result = succ( expression )
```

`expression`	ordinal
	spatial
`Result`	ordinal
	spatial

Operation

The result of the operation depends on wheter `expression` has a legend or not. If `expression` has a legend, the legend determines the domain of `expression`: the domain consists of the ordinal numbers linked to the classes in the legend. This domain with these ordinal classes are also assigned to `Result`. Cells on `Result` may have values in this domain. For each `expression` cell value the first higher ordinal number which is in the domain is determined. This is assigned to the corresponding cell on `Result`.

If `expression` does not have a legend an ordinal number is assigned to `Result` which is the ordinal number on `expression` plus 1, on a cell- by-cell basis.

Notes

A cell on `expression` with missing value is assigned a missing value on `Result`.

Group

This operation belongs to the group of Data management

Examples

1.

```
pcrcalc  Result1.map = succ(Expr.map)
```

Result1.map

-4	10	10	1
-4	-4	10	1
-4	10	10	3
5	5	10	MV

Expr.map

-5	9	9	0
-5	-5	9	0
-5	9	9	2
4	4	9	MV

tan

Name

tan -- Tangent

Synopsis

pcrcalc [option] Result = **tan**(expression)

expression	directional, scalar
	spatial, non spatial
Result	scalar
	dimension of expression

Options

--degrees or --radians

--degrees

if expression is a number then the unit is degrees (default)

--radians

if expression is a number then the unit is radians

Operation

For each cell, calculates the tangent of expression cell value and assigns it to Result.

Notes

A cell with missing value on expression is assigned a missing value on Result.

If expression is of directional data type, a cell on expression without a direction (cell value -1) is assigned a missing value.

Group

This operation belongs to the group of Arithmetic operators

Examples

1.

```
pcrcalc  Result.map = tan(Expr.map)
```

Result.map

MV	0.577	1
0.0660	0.0349	0
0.0559	-0.51	MV

Expr.map

MV	30	45
3.8	2	0
3.2	153	90

time

Name

time -- Timestep

Synopsis

pcrcalc `Result = ` **time** `()`

`Result` scalar

non spatial

Operation

This operation is used in the iterative sections (dynamic, storage and transport sections) of a dynamic model script only. For each timestep in a model run, the operator assigns to `Result` the time *t* at the timestep *i* under consideration. The time at the first timestep (*i* = 1) is *t*(start), the time at the second timestep (*i* = 2) is *t*(start) + *timeslice*, at the third time step (*i* = 3) is *t*(start) + 2 x *timeslice*, etc. The time dimension in a model (i.a. *t*(start), *timeslice*) is defined in the timer section of a Dynamic Modelling Script.

Group

This operation belongs to the group of Time operators

Examples

1.

```
pcrcalc  Result.map = time()
```

Result.map

1	0	1
MV	1	1
1	1	0

Prev tan | Home | Up | Next timeinput...

timeinput...

Name

timeinput... -- Cell values per timestep read from a time series that is linked to a map with unique identifiers

Synopsis

```
pcrcalc Result = timeinput... type( TimeSeries, idexpression )
pcrcalc Result = timeinputnominal( TimeSeries, idexpression )
pcrcalc Result = timeinputboolean( TimeSeries, idexpression )
pcrcalc Result = timeinputnominal( TimeSeries, idexpression )
pcrcalc Result = timeinputordinal( TimeSeries, idexpression )
pcrcalc Result = timeinputscalar( TimeSeries, idexpression )
pcrcalc Result = timeinputdirectional( TimeSeries, idexpression )
pcrcalc Result = timeinputldd( TimeSeries, idexpression )
```

`TimeSeries`	ascii formatted time series
`idexpression`	boolean, nominal, ordinal
	spatial, non spatial
`Result`	type is specified by the sort of command: **timeinputboolean** results in a boolean `Result`, **timeinputnominal** in a nominal `Result` etc.
	spatial

Operation

Example 1. Example of a time series file with header

rain (mm) per area for model with timer 1 6 1
4
model time
Area station A
Area station B

Area station C
1 2.0 0.0 0.0
2 3.0 2.0 1.0
3 7.0 5.0 3.0
4 9.0 12.0 6.0
5 6.0 0.0 5.0
6 0.0 1.0 2.0

This operation is used in the iterative sections (dynamic, storage and transport sections) of a dynamic model script only. `TimeSeries` is an ascii formatted time series that contains cell values formatted in rows and columns. During a run of a Dynamic Model `TimeSeries` is read from top to bottom: for each timestep, a row gives cell values that are assigned to `Result` at the timestep under consideration. The cell values in a row are assigned to the cells of `Result` on basis of unique identifiers on `idexpression`: each column on `TimeSeries` gives cell values for an unique identifier. Each timestep, the cell value in a column is assigned to the cells on `Result` that have a unique identifier on `idexpression` that corresponds with the unique identifier of the column.

The data type that is assigned to `Result` is specified by the sort of operator that is used.

The contents and partly also the format (number of rows) of the `TimeSeries` must match the dynamic model for which the `TimeSeries` is used, especially the time dimension of the model. For a description of the time dimension and the terms used, see section VI.2.2d. Two types of format for the `TimeSeries` can be used; the **timeinput...** operator detects the formats by itself:

1) a time series file with a header

line 1: header, description

line 2: header, number of columns in the file

line 3: header, time column description

line 4 up to and including line $n + 3$: header, the names of the n unique identifiers.

subsequent lines: data formatted in rows and columns. Each row represents one timestep i at time $t(i)$ in the model for which the time series is used; the first row contains data for timestep $i = 1$, the second row for timestep $i = 2$, etc. The first column contains the time t at the timesteps. The remaining columns (column number 2 and further) contain values that are assigned to `Result`. These values must be in the domain of the data type that is given to `Result`. Each column contains data for an unique identifier. Column number 2 is associated with an unique identifier 1, column number 3 with an unique identifier 2, etc. In general, starting with the second column, a column number c is related to an unique identifier $c - 1$. The columns must be separated by one or more whitespace characters (spaces, tabs), the number of characters does not matter.

2) a plain time series file

This is a file formatted like the time series file with header, but without header lines.

During operation, the values in a column of `TimeSeries` are assigned to the cells of `Result` that have an unique identifier on `idexpression` that corresponds with the unique identifier value associated

with the column of `TimeSeries`. So, `idexpression` must contain a set of whole unique identifier cell values, starting with a value one, that is related to the unique identifiers of the columns in `TimeSeries`: cells with a value 1 are assigned data from the second column in `TimeSeries`, cells with a value 2 are assigned data from the third column etc.

An example of a time series file with header is given in the table explained above. It is meant for a dynamic model with starttime 1, endtime 6 and timeslice 1. It gives precipitation for three areas. For instance, values for the area of station B are in the third column. These are assigned to cells of `Result` that have a value 2 on `idexpression`.

Notes

Cells with an unique identifier on `idexpression` that is not represented by a column associated with the same unique identifier on `TimeSeries` are assigned a missing value on `Result`. For instance: let `TimeSeries` contain three columns, the first column with the time and two columns with data associated with unique identifiers 1 and 2 respectively. All cells of `Result` that have a `idexpression` value different from 1 or 2 are assigned a missing value.

A timeseries generated and stored in the database during a model run by the report keyword (or the **timeoutput** operator) cannot be imported during the same model run with the **timeinput...** operator.

Group

This operation belongs to the group of Time operators

Prev time | Home Up | Next timeinput

timeinput

Name

timeinput -- Set of output maps per timestep with an extension that refers to the time at the timestep

Synopsis

pcrcalc `Result = ` **timeinput**`( suffixOfMap )`

`suffixOfMap`	boolean, nominal, ordinal, scalar, directional, ldd spatial, non spatial
`Result`	type of `suffixOfMap` spatial

Operation

This operation is used in the iterative sections (dynamic, storage and transport sections) of a dynamic model script only. For each timestep **timeinput** assigns to `Result` one map of a set of sequential maps given by `suffixOfMap`.

`suffixOfMap` refers to a set of maps that all have a filename starting with the suffix `suffixOfMap`. Additionally these maps contain a unique time extension Ext in their file name, directly after `suffixOfMap`. For each map, this time extension refers to the time at a timestep in a model run: each timestep, `Result` is assigned the `suffixOfMap`Ext with the time extension Ext that corresponds with the time *t*(*i*) at the timestep *i* under consideration.

For each timestep *i* a `suffixOfMap`Ext must be available in the PCRaster database with a time extension Ext corresponding with the time *t*(*i*) at the timestep under consideration. These maps referred to as `suffixOfMap` must have the following filenames. The filenames consist of 8 characters, a dot and 3 characters. This is in accordance with the ordinary rules for filenames in DOS. Each filename starts with the name of the suffix `suffixOfMap`. This suffix name may be maximal 8 characters long. All the remaining character positions must be used for the time extension, which is unique and different for each map. The time extension must be a whole value; the map will be assigned to `Result` at timestep *i* with time *t*(*i*) = Ext.

Two examples are given to illustrate the use of the operator. Imagine a model with startime = 4, endtime 10 and a timeslice of 2. As a result, this model consists of 4 timesteps at time 4, 6, 8, 10. During a model run, the operation **timeinput**(Rain) queries for maps in the PCRaster database with filenames:

Rain0000.004, Rain0000.006, Rain0000.008 and Rain0000.010. These maps are assigned to `Result` at the sequential timesteps at time 4, 6, 8, 10.

In a model with starttime 990, endtime 1010 and a timeslice of 10, the operation **timeinput**(Water) will query for Water000.990, Water001.000 and Water001.010.

Notes

Maps that are reported in a model with the report keyword are stored in the database with a filename format that corresponds with the format needed for the **timeinput** operation. So, one model may be used to generate a set of suffix maps; these maps can be used in another model as input maps to a **timeinput** operation.

A stack of maps generated and stored in the database during a model run by the report keyword cannot be imported during the same model run with the **timeinput** operator.

Group

This operation belongs to the group of Time operators

Examples

1.

```
pcrcalc  Result.map = time()
```

`Result.map`

1	0	1
MV	1	1
1	1	0

timeoutput

Name

timeoutput -- Expression value of an uniquely identified cell or cells written to a time series per timestep

Synopsis

pcrcalc ResultTimeSeries = **timeoutput**(idexpression, expression)

idexpression	boolean, nominal, ordinal
	spatial
expression	boolean, nominal, ordinal, scalar, directional, ldd
	spatial
ResultTimeSeries	time series

Operation

This operation is used in the iterative sections (dynamic, storage and transport sections) of a dynamic model script only. The keyword report precedes the operation. The idexpression is an expression that contains one identified cell or several uniquely identified cells. For each timestep the value of expression at the identified cell or cells is written to ResultTimeSeries, which is an ascii formatted time series file. After a model run, the time series contains for each identified cell a list of expression cell values per timestep.

The idexpression must contain one or more uniquely identified cells, which are numbered with consecutive whole values, starting with 1. The remaining cells must have a value 0. For instance, if you want to write the expression values from three different cells to ResultTimeSeries, these cells must have the values 1, 2 and 3 on idexpression respectively, the remaining cells must have a value 0.

The ResultTimeSeries is an ascii formatted time series with header. It has the following lay out:

line 1: header, description: expression map name

line 2: header, number of columns in the file

line 3: header, time column description: model time

line 4 up to and including line *n* + 3: header, the numbers of the *n* uniquely identified cells: 1,2,3,...*n*.

subsequent lines: data formatted in rows and columns. Each row represents one timestep *i* at time *t*(*i*) in the model from which the time series is the result; the first row contains data for timestep *i* = 1, the second row for timestep *i* =2, etc. The first column contains the time *t* at the timesteps. At the first row which contains data for the first time step (*i* = 1) it is always the starttime *t*(1). The remaining columns (column number 2 up to and including *n* + 1, where *n* is the number of uniquely identified cells as above said) contain values that are taken from `expression`: column number *n* + 1 contains data of the cell that has a value *n*.

Notes

In principle, each unique identifier is represented by one cell on `idexpression`. This is the basic use: to sample certain locations. Alternatively, more than one cell on `idexpression` may have the same unique identifier value. In that case `ResultTimeSeries` contains for the id under consideration an aggregate `expression` value of the set of cells with that id. If `expression` is of data type scalar or directional the average `expression` value of cells with the id under consideration on `idexpression` is written to `ResultTimeSeries`. If the data type is directional cells without a direction (cellvalue -1) are discarded in the calculation of the average value. If all cells with the id under consideration have no direction, a -1 value is written to `ResultTimeSeries`. If `expression` is of data type boolean, nominal or ordinal the highest score (most ocurring cell value) of the cells with the id under consideration is written to the time series. If several values all have the highest score, the highest value is assigned.

Group

This operation belongs to the group of Time operators

Examples

1.

```
pcrcalc  Result.map = timeslice()
```

`Result.map`

1	0	1
MV	1	1
1	1	0

timeslice

Name

timeslice -- Timeslice

Synopsis

pcrcalc Result = **timeslice** ()

Result scalar

non spatial

Operation

This operation is used in a dynamic model script only. It assigns the timeslice used in the model. The timeslice is specified in the timer section of the script. The timeslice is always 1.

Group

This operation belongs to the group of Time operators

Examples

1.

```
pcrcalc  Result.map = timeslice()
```

Result.map

1	0	1
MV	1	1
1	1	0

timeoutput Up uniform

uniform

Name

uniform -- Boolean TRUE cell gets value from an uniform distribution

Synopsis

pcrcalc Result = **uniform**(expression)

expression	boolean
	spatial, non spatial
Result	scalar
	spatial

Operation

A random generator is used to generate the `Result`: for each cell that has a value 1 (TRUE) on `expression`, a value is taken from a uniform distribution between 0 and 1 and assigned to the cell on `Result`. Cells that have a value 0 (FALSE) on `expression` area assigned a missing value.

Notes

Group

This operation belongs to the group of Random number generators; Cells

See Also

Section 6.3.2 Section 6.4.2

Examples

1.
```
pcrcalc  Result.map = uniform(uniqueid.map)
```

Result.map

MV	MV	MV	MV	MV
0.855	0.272	0.283	0.814	0.305
0.621	0.49	0.83	0.827	0.37
MV	1	0.436	0.733	0.398
MV	0.143	0.183	0.583	0.715

Expr.map

MV	0	MV	MV	0
1	1	1	1	1
1	1	1	1	1
0	1	1	1	1
0	1	1	1	1

Prev timeslice | Home Up | Next uniqueid

uniqueid

Name

uniqueid -- Unique whole value for each Boolean TRUE cell

Synopsis

pcrcalc Result = **uniqueid**(expression)

expression	boolean
	spatial, non spatial
Result	scalar
	spatial

Operation

For each cell that has a value 1 (TRUE) on `expression` assigns a unique whole positive value to `Result`, starting with 1. Cells that have a value 0 (FALSE) on `expression` are assigned a value 0.

Notes

A cell with a missing value on `expression` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Coordinates, unique ID's

Examples

1.
```
pcrcalc  Result.map = uniqueid(Expr.map)
```

`Result.map`

MV	0	MV	MV	0
1	2	3	4	5
6	7	8	9	10
0	11	12	13	14
0	15	16	17	18

`Expr.map`

MV	0	MV	MV	0
1	1	1	1	1
1	1	1	1	1
0	1	1	1	1
0	1	1	1	1

Prev uniform | Home | Up | Next upstream

upstream

Name

upstream -- Sum of the cell values of its first upstream cell(s)

Synopsis

pcrcalc Result = **upstream**(ldd, material)

ldd	ldd
	spatial
material	scalar
	spatial, non spatial
Result	scalar
	spatial

Operation

For each cell the neighbour cells that have a local drain direction on `ldd` towards the cell are determined. These are cells that drain directly to the cell. On `Result` the cell is assigned the sum of the `material` values of these first upstream cells. This is done for each cell.

Notes

A cell with a missing value on `ldd` or `material` is assigned a missing value on `Result`. Additionally the downstream neighbour of a missing value cell on `material` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Neighbourhood operators; local drain directions

Examples

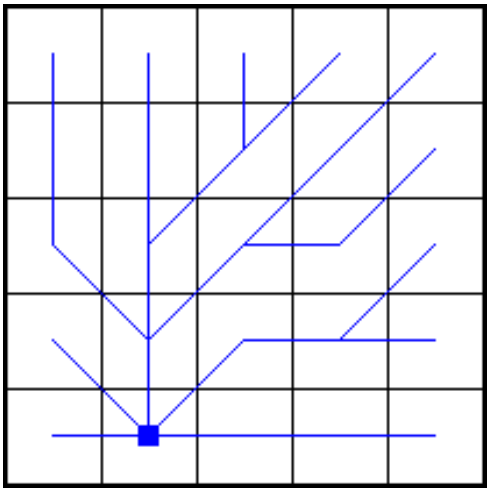
1.
```
pcrcalc  Result.map=upstream(Ldd.map, Expr.map)
```

upstream

Result.map

0	0	0	0	0
1	1	4	4	0
1	3	MV	4	0
0	MV	4	8	0
0	14	5	5	0

Ldd.map



Expr.map

1	1	2	2	4
1	1	2	2	4
2	2	MV	4	4
2	2	2	4	4
3	7	5	5	5

Prev uniqueid | Home Up | Next view

view

Name

view -- TRUE or FALSE value for visibility from viewpoint(s) defined by a digital elevation model

Synopsis

pcrcalc Result = **view**(elevation, viewpoints)

elevation	scalar
	spatial
viewpoints	boolean
	spatial
Result	boolean
	spatial

Operation

All cells in `viewpoints` with a value 1 (TRUE) are used as viewpoints. The 3D landscape over which is looked out is defined by the elevation model `elevation`. Each cell which is visible from the TRUE cells on `viewpoints` is assigned a 1 (TRUE) on `Result`. Cells which are not visible are assigned a 0 (FALSE) on `Result`.

Notes

A cell with a missing value on `elevation` is assigned a missing value on `Result`. If `elevation` has a missing value on the line of sight between a cell and its viewpoint from which it can potentially be seen, the cell is considered as non visible from that viewpoint. If there isn't a different viewpoint from which the cell can be seen, the cell is assigned a 0 (FALSE) on `Result`.

Group

This operation belongs to the group of Neigbourhood operator; operators for visibility analysis

Examples

1.
```
pcrcalc  Result.map = view(Dem.map, Points.map)
```

view

Result.map

1	1	1	0	0
1	1	1	1	0
1	1	1	1	1
1	MV	1	0	1
1	0	1	1	1

Dem.map

-10	10	20	-8	-5
5	5	-2	22	-3
6	6	0	0	0
7	8	2	-21	13
8	10	10	20	60

Points.map

0	1	0	0	0
0	0	0	0	0
0	0	0	0	0
0	MV	0	0	0
0	0	0	0	0

Prev upstream | Home Up | Next windowaverage

windowaverage

Name

windowaverage -- Average of cell values within a specified square neighbourhood

Synopsis

pcrcalc [option] Result = **windowaverage**(expression, windowlength)

expression	scalar
	spatial
windowlength	scalar
	spatial, non spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell

--unittrue

windowlength is measured in true length (default)

--unitcell

windowlength is measured in number cell lengths

Operation

For each cell its windowaverage is computed as follows. A square window with the cell in its centre is defined by windowlength. The windowlength is the length of the window in horizontal and vertical directions. For each cell *i* which is entirely or partly in the window the fraction of the cell in the window is determined. This is the area of the part of the cell in the window divided by the total area of a cell. Call this *fraction*(*i*); let expression(*i*) be the expression value of cell *i*. Now, the **windowaverage**(*c*) of the centercell is computed by taking a fraction weighted average of the values on expression:

windowavaverage(c) =``` {big sum from i=1 to n {lpar @Quote {"fraction"} (i) times expression(i) rpar }}over {{big sum from i=1 to n {@Quote {"fraction"} (i)}}}

where *n* is the number of cells which is entirely or partly in the window. For each cell *c* its windowaverage is computed and assigned to the corresponding cell on `Result`.

Notes

The cell value on `windowlength` should be greater than 0, else a missing value is assigned to the corresponding cell on `Result`.

A cell on `windowlength` with a missing value results in a missing value on `Result` at the corresponding cell. However, if a missing value on `windowlength` occurs in a cell which is not the centre cell of the window the `expression` value in that cell *is* included in the summation of the cell values in the window.

Group

This operation belongs to the group of Neigbourhood operators; window operators

Examples

1.

```
pcrcalc  Result1.map = windowaverage( Expr.map, 6)
```

`Result1.map`

0.333	0.6	-5.4	-4.83	-7.75
1.2	1.38	-2.25	-2.22	-4.17
1.4	1.62	6.75	5.89	7.83
0.667	1.44	6.67	7.22	9.33
-0.25	0.833	8.5	9.33	12.5

`Expr.map`

0	-1	1	-30	0
2	MV	1	2	-3
3	2	3	4	2
0	0	2	40	2
1	-2	4	7	1

2.

```
pcrcalc  Result2.map = windowaverage( Expr.map, WinLen2.map)
```

Result2.map

0	-1	1	-30	0
1.5	1.25	-0.643	-2.62	-4.17
1.82	1.79	6.75	5.89	5.88
0.667	2.38	5.84	2.75	3.3
-0.25	2.3	7.19	5.42	7.22

Expr.map

0	-1	1	-30	0
2	MV	1	2	-3
3	2	3	4	2
0	0	2	40	2
1	-2	4	7	1

WinLen2.map

2	2	2	2	2
4	4	4	4	6
4	4	6	6	6.8
6	6.8	6.8	12	12
6	6.8	6.8	10	10

Prev view | Home | Up | Next windowdiversity

windowdiversity

Name

windowdiversity -- Number of unique values within a specified square neighbourhood

Synopsis

pcrcalc [option] Result = **windowdiversity**(expression, windowlength)

expression	boolean, nominal, ordinal
	spatial
windowlength	scalar
	spatial, non spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell

--unittrue

windowlength is measured in true length (default)

--unitcell

windowlength is measured in number of cell lengths

Operation

For each cell a square window with the cell in its centre is defined by windowlength. The windowlength is the length of the window in horizontal and vertical directions. For each cell on expression the number of unique values within its window is counted. This number is assigned to the corresponding cell on Result. Both cells on expression which are entirely in the window and cells which are partly in the window are considered.

Notes

The cell value on windowlength should be greater than 0, else a missing value is assigned to the corresponding cell on Result.

A cell on windowlength with a missing value results in a missing value on Result at the corresponding cell. However, if a missing value on windowlength occurs in the window in a cell which is not the centre cell of the window the expression value in that cell *is* considered for determination of the number of unique values in the window.

Group

This operation belongs to the group of Neigbourhood operators; window operators

Examples

1.

```
pcrcalc  Result1.map = windowdiversity( Expr.map, 6)
```

Result1.map

2	2	3	3	1
4	4	5	4	2
4	4	4	3	2
6	6	5	3	3
4	5	4	3	3

Expr.map

1	1	1	-4	-4
2	MV	2	-4	-4
3	6	3	3	3
4	2	3	3	3
0	14	3	-1	0

2.

```
pcrcalc  Result2.map = windowdiversity( Expr.map, WinLen2.map)
```

Result2.map

1	1	1	1	1
4	4	5	4	2
4	4	4	3	6
6	8	8	9	8
4	7	7	6	3

Expr.map

1	1	1	-4	-4
2	MV	2	-4	-4
3	6	3	3	3
4	2	3	3	3
0	14	3	-1	0

WinLen2.map

2	2	2	2	2
4	4	4	4	6
4	4	6	6	6.8
6	6.8	6.8	12	12
6	6.8	6.8	10	10

3.

```
pcrcalc  Result3.omap = windowdiversity(Expr2.imap,celllength() * 1.1)) gt 1)
```

Result3.map

0	0	0	0	0	0
0	0	1	1	1	0
1	1	1	1	1	1
1	1	1	1	1	1
1	1	0	0	0	0
1	1	0	0	0	0

Expr2.map

2	2	2	2	2	2
2	2	2	2	2	2
2	2	2	1	2	2
2	1	1	1	1	1
2	1	1	1	1	1
1	1	1	1	1	1

Prev windowaverage | Home Up | Next windowhighpass

windowhighpass

Name

windowhighpass -- Increases spatial frequency within a specified square neighbourhood

Synopsis

pcrcalc [option] Result = **windowhighpass**(expression, windowlength)

expression	scalar
	spatial
windowlength	scalar
	spatial, non spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell

--unittrue

windowlength is measured in true length (default)

--unitcell

windowlength is measured in number of cell lengths

Operation

For each cell *c* its windowhighpass value is computed as follows. A square window with the cell *c* in its centre is defined by windowlength. The windowlength is the length of the window in horizontal and vertical directions. For each cell *i* which is entirely or partly in the window and which is not the centre cell *c* the fraction of the cell in the window is determined. This is the area of the part of the cell in the window divided by the total area of a cell. Let *fraction*(*i*) be this fraction; let expression(*i*) be the expression value of a surrounding cell *i* and expression(*c*) the expression value of the centre cell *c*. The windowhighpass filter value on the centre cell *c* is calculated according to:

highpass(c) =``` 2 times expression(c) times {big sum from i=1 to n {@Quote {"fraction"} (i)}````` - `````big sum from i=1 to n lbrace{expression(i) times @Quote {"fraction"} (i)}rbrace where *n* is the number of cells *i* surrounding the centre cell *c* in the window. This computation is performed for all cells: for each cell the highpass(*c*) value is assigned to the corresponding cell on Result.

Notes

The cell value on `windowlength` should be greater than 0, else a missing value is assigned to the corresponding cell on `Result`.

A cell on `windowlength` with a missing value results in a missing value on `Result` at the corresponding cell. However, if a missing value on `windowlength` occurs in a cell which is not the centre cell of the window the `expression` value in that cell *is* included in the computation of the highpass value in the window.

Group

This operation belongs to the group of Neigbourhood operators; window operators

Examples

1.
```
pcrcalc  Result1.map = windowhighpass( Expr.map, 6)
```

`Result1.map`

-1	-12	36	-301	31
12	MV	33	54	-8
20	17	-9	15	-25
-4	-13	-26	615	-34
8	-27	-7	21	-43

`Expr.map`

0	-1	1	-30	0
2	MV	1	2	-3
3	2	3	4	2
0	0	2	40	2
1	-2	4	7	1

2.
```
pcrcalc  Result2.map = windowhighpass( Expr.map, WinLen2.map)
```

`Result2.map`

MV	MV	MV	MV	MV
4.25	MV	8.25	24.5	-8
8.5	5.75	-9	15	-16.1
-4	-23.9	-21.5	6.45e+08	3.75
8	-42.3	2.04	96	-48

`Expr.map`

0	-1	1	-30	0
2	MV	1	2	-3
3	2	3	4	2
0	0	2	40	2
1	-2	4	7	1

`WinLen2.map`

2	2	2	2	2
4	4	4	4	6
4	4	6	6	6.8
6	6.8	6.8	12	12
6	6.8	6.8	10	10

windowmajority

Name

windowmajority -- Most occurring cell value within a specified square neighbourhood

Synopsis

pcrcalc [option] Result = **windowmajority**(expression, windowlength)

expression	boolean, nominal, ordinal
	spatial
windowlength	scalar
	spatial, non spatial
Result	type of expression
	spatial

Options

--unittrue or -unitcell

--unitcell

windowlength is measured in true length (default)

--unitcell

windowlength is measured in number of cell lengths

Operation

For each cell a square window with the cell in its centre is defined by windowlength. The windowlength is the length of the window in horizontal and vertical directions. For each cell on expression, the most often occurring cell value within its window is determined and assigned to the corresponding cell on Result. Both cells on expression which are entirely in the window and cells which are partly in the window are considered. At a cell, if two or more values occur the same (largest) number of times in its window, the windowlength at that cell is increased with two times the celllength of expression. The most often occurring cell value in this enlarged window is assigned to the cell under consideration. If this enlarged window still does not result in one most often occurring value, the window is progressively enlarged (with steps of two times the cellsize of expression) until a most often occuring cell value is found in the window.

Notes

The cell value on windowlength should be greater than 0, else a missing value is assigned to the corresponding cell on Result.

A cell on `windowlength` with a missing value results in a missing value on `Result` at the corresponding cell. However, if a missing value on `windowlength` occurs which is not the centre cell of the window the value on `expression` in that cell *is* included in the computation of the average of the cell values in the window.

Group

This operation belongs to the group of Neigbourhood operators; window operators

Examples

1.
```
pcrcalc  Result1.map = windowmajority( Expr.map, 6)
```

`Result1.map`

17	17	2	4	3
17	17	2	15	3
15	15	15	15	15
2	15	15	15	15
15	15	15	15	15

`Expr.map`

17	MV	4	MV	3
17	MV	2	2	3
17	8	-7	6	4
0	0	15	15	15
2	2	15	15	15

2.
```
pcrcalc  Result2.map = windowmajority( Expr.map, WinLen2.map)
```

`Result2.map`

17	MV	4	MV	3
17	17	2	2	3
17	8	15	15	15
2	15	15	15	15
15	15	15	15	15

`Expr.map`

17	MV	4	MV	3
17	MV	2	2	3
17	8	-7	6	4
0	0	15	15	15
2	2	15	15	15

`WinLen2.map`

2	2	2	2	2
4	4	4	4	6
4	4	6	6	6.8
6	6.8	6.8	12	12
6	6.8	6.8	10	10

Prev windowhighpass | Home | Up | Next windowmaximum

windowmaximum

Name

windowmaximum -- Maximum cell value within a specified square neighbourhood

Synopsis

pcrcalc [option] Result = **windowmaximum**(expression, windowlength)

expression	ordinal, scalar
	spatial
windowlength	scalar
	spatial, non spatial
Result	expression
	spatial

Options

--unittrue or --unitcell

--unittrue

windowlength is measured in true length (default)

--unitcell

windowlength is measured in number of cell lengths

Operation

For each cell a square window with the cell in its centre is defined by windowlength. The windowlength is the length of the window in horizontal and vertical directions. For each cell on expression, the maximum cell value within its window is determined and assigned to the corresponding cell on Result. Both cells on expression which are entirely in the window and cells which are partly in the window are considered.

Notes

The cell value on windowlength should be greater than 0, else a missing value is assigned to the corresponding cell on Result.

A cell on windowlength with a missing value results in a missing value on Result at the corresponding cell. However, if a missing value on windowlength occurs in a cell in the window which is not the centre cell of the window the expression value in that cell is considered for determination of the maximum cell value in the window.

Group

This operation belongs to the group of Neigbourhood operators; window operators

Examples

1.

```
pcrcalc  Result1.map = windowmaximum(Expr.map, 6)
```

Result1.map

2	2	2	2	2
3	3	4	4	4
3	3	40	40	40
3	4	40	40	40
1	4	40	40	40

Expr.map

0	-1	1	-30	0
2	MV	1	2	-3
3	2	3	4	2
0	0	2	40	2
1	-2	4	7	1

2.

```
pcrcalc  Result2.map= windowmaximum( Expr.map, WinLen2.map)
```

Result2.map

0	-1	1	-30	0
3	3	4	4	4
3	3	40	40	40
3	40	40	40	40
1	40	40	40	40

Expr.map

0	-1	1	-30	0
2	MV	1	2	-3
3	2	3	4	2
0	0	2	40	2
1	-2	4	7	1

WinLen2.map

2	2	2	2	2
4	4	4	4	6
4	4	6	6	6.8
6	6.8	6.8	12	12
6	6.8	6.8	10	10

Prev windowmajority | Home | Up | Next windowminimum

windowminimum

Name

windowminimum -- Minimum value within a specified square neighbourhood

Synopsis

pcrcalc [option] Result = **windowminimum**(expression, windowlength)

expression	ordinal, scalar
	spatial
windowlength	scalar
	spatial, non spatial
Result	expression
	spatial

Options

--unittrue or --unitcell

--unittrue

windowlength is measured in true length (default)

--unitcell

windowlength is measured in number of cell lengths

Operation

For each cell a square window with the cell in its centre is defined by windowlength. The windowlength is the length of the window in horizontal and vertical directions. For each cell on expression, the minimum cell value within its window is determined and assigned to the corresponding cell on Result. Both cells on expression which are entirely in the window and cells which are partly in the window are considered.

Notes

The cell value on windowlength should be greater than 0, else a missing value is assigned to the corresponding cell on Result.

A cell on windowlength with a missing value results in a missing value on Result at the corresponding cell. However, if a missing value on windowlength occurs in a cell in the window which is not the centre cell of the window the value on expression in that cell *is* considered for determination of the minimum cell value in the window.

Group

This operation belongs to the group of Neigbourhood operators; window operators

Examples

1.

```
pcrcalc  Result1.map= windowminimum( Expr.map, 6)
```

Result1.map

-1	-1	-30	-30	-30
-1	-1	-30	-30	-30
0	0	0	-3	-3
-2	-2	-2	1	1
-2	-2	-2	1	1

Expr.map

0	-1	1	-30	0
2	MV	1	2	-3
3	2	3	4	2
0	0	2	40	2
1	-2	4	7	1

2.

```
pcrcalc  Result2.map=windowminimum(Expr.map, WinLen2.map)
```

Result2.map

0	-1	1	-30	0
-1	-1	-30	-30	-30
0	0	0	-3	-30
-2	-2	-3	-30	-30
-2	-2	-2	-2	1

Expr.map

0	-1	1	-30	0
2	MV	1	2	-3
3	2	3	4	2
0	0	2	40	2
1	-2	4	7	1

WinLen2.map

2	2	2	2	2
4	4	4	4	6
4	4	6	6	6.8
6	6.8	6.8	12	12
6	6.8	6.8	10	10

Prev windowmaximum | Home | Up | Next windowtotal

windowtotal

Name

windowtotal -- Sum of values within a specified square neighbourhood

Synopsis

pcrcalc [option] Result = **windowtotal**(expression, windowlength)

expression	scalar
	spatial
windowlength	scalar
	spatial, non spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell

--unitcell

windowlength is measured in true length (default)

--unitcell

windowlength is measured in number of cell lengths

Operation

For each cell its windowtotal is computed as follows. A square window with the cell in its centre is defined by windowlength. The windowlength is the length of the window in horizontal and vertical directions. For each cell *i* which is entirely or partly in the window the fraction of the cell in the window is determined. This is the area of the part of the cell in the window divided by the total area of a cell. Call this *fraction*(*i*); let expression(*i*) be the value on expression of cell *i*. Now, the windowtotal of the centre cell *c* is a fraction weighted sum of the values on expression:

windowtotal(c) =``` {big sum from i=1 to n {lpar @Quote {"fraction"} (i) times expression(i) rpar}}

where *n* is the number of cells which is entirely or partly in the window. For each cell *c* its windowtotal is computed and assigned to the corresponding cell on Result.

Notes

The cell value on `windowlength` should be greater than 0, else a missing value is assigned to the corresponding cell on `Result`.

A cell on `windowlength` with a missing value results in a missing value on `Result` at the corresponding cell. However, if a missing value on `windowlength` occurs in a cell which is not the centre cell of the window the value on `expression` in that cell *is* included in the summation of the cell values in the window.

Group

This operation belongs to the group of Neigbourhood operators; window operators

Examples

1. `pcrcalc  Result1.map = windowtotal( Expr.map, 6)`

   `Result1.map`

1	3	-27	-29	-31
6	11	-18	-20	-25
7	13	54	53	47
4	13	60	65	56
-1	5	51	56	50

   `Expr.map`

0	-1	1	-30	0
2	MV	1	2	-3
3	2	3	4	2
0	0	2	40	2
1	-2	4	7	1

2. `pcrcalc  Result2.map = windowtotal( Expr.map, WinLen2.map)`

   `Result2.map`

0	-1	1	-30	0
3.75	3.75	-2.25	-10.5	-25
5	6.25	54	53	44
4	23.9	62.4	53	50.2
-1	16.2	53.8	65	65

   `Expr.map`

0	-1	1	-30	0
2	MV	1	2	-3
3	2	3	4	2
0	0	2	40	2
1	-2	4	7	1

   `WinLen2.map`

2	2	2	2	2
4	4	4	4	6
4	4	6	6	6.8
6	6.8	6.8	12	12
6	6.8	6.8	10	10

xcoordinate

Name

xcoordinate -- X-coordinate of each Boolean TRUE cell

Synopsis

pcrcalc [option] Result = **xcoordinate**(expression)

expression	boolean
	spatial, non spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell

--unittrue

coordinates are expressed in true distance coordinates (default)

--unitcell

coordinates are expressed in unit cell lengths, where the minimum x coordinate of a cell centre is 0.5 and the maximum x coordinate of a cell centre is x - 0.5, where x is the number of columns of cells.

assignment of coordinates

--coorcentre

for each cell, the coordinate of the cell centre is assigned (default)

--coorul

for each cell, the coordinate of the upper left corner is assigned

--coorlr

for each cell, the coordinate of the lower right corner is assigned

Operation

For each cell that has a value 1 (TRUE) on `expression` assigns the x coordinate of the cell to the cell on `Result`. Cells with a value 0 (FALSE) on `expression` are assigned a missing value.

Notes

A cell with a missing value on `expression` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Coordinates, unique ID's

Examples

1.

```
pcrcalc  --coorlr Result.map = xcoordinate(Expr.map)
```

`Result.map`

MV	MV	MV	MV	MV
MV	20	25	30	MV
15	20	25	30	35
15	20	25	30	35
15	20	25	MV	35

`Expr.map`

MV	MV	0	0	0
MV	1	1	1	0
1	1	1	1	1
1	1	1	1	1
1	1	1	0	1

xor

Name

xor -- Boolean-XOR operation

Synopsis

pcrcalc Result = expression1 **xor** expression2

expression1	boolean
	spatial, non spatial
expression2	boolean
	spatial, non spatial
Result	boolean
	spatial; non spatial if expression1 and expression2 are non spatial

Operation

The cell values on expression1 and expression2 are interpreted as Boolean values; where 1 is TRUE and 0 is FALSE. For each cell the Boolean XOR evaluation is performed: if both expression1 and expression2 have a cell value 1 (TRUE) or both have a cell value 0 (FALSE) Result has a cell value 0 (FALSE) on the corresponding cell; if one of the expressions (expression1 or expression2) has a cell value 1 (TRUE) and the other a cell value 0 (FALSE) Result has cell value 1 (TRUE).

Table 1. Cross table of the XOR operator.

XOR		**expression1**	
		False	True
expression2	False	False	True
	True	True	False

Notes

A cell with missing value on `expression1` or `expression2` or on both expressions results in a missing value on `Result` at the corresponding cell.

Group

This operation belongs to the group of Boolean operators

Examples

1.

```
pcrcalc  Result.map = Expr1.map xor Expr2.map
```

`Result.map`

1	0	0
1	MV	MV
0	0	1

`Expr1.map`

1	1	0
0	MV	0
1	1	0

`Expr2.map`

0	1	0
1	1	MV
1	1	1

Prev xcoordinate | Home | Up | Next ycoordinate

ycoordinate

Name

ycoordinate -- Y-coordinate of each Boolean TRUE cell

Synopsis

pcrcalc [option] Result = **ycoordinate**(expression)

expression	boolean
	spatial, non spatial
Result	scalar
	spatial

Options

--unittrue or --unitcell

--unittrue

coordinates are expressed in true distance coordinates (default)

--unitcell

coordinates are expressed in unit cell lengths, where the minimum y coordinate of a cell centre is 0.5 and the maximum y coordinate of a cell centre is y - 0.5, where y is the number of rows of cells.

assignment of coordinates

--coorcentre

for each cell, the coordinate of the cell centre is assigned (default)

--coorul

for each cell, the coordinate of the upper left corner is assigned

--coorlr

for each cell, the coordinate of the lower right corner is assigned

Operation

For each cell that has a value 1 (TRUE) on `expression` assigns the y coordinate of the cell to the cell on `Result`. Cells with a value 0 (FALSE) on `expression` are assigned a missing value.

Notes

A cell with a missing value on `expression` is assigned a missing value on `Result`.

Group

This operation belongs to the group of Coordinates, unique ID's

Examples

1.

```
pcrcalc  --coorlr Result.map = ycoordinate(Expr.map)
```

`Result.map`

MV	MV	MV	MV	MV
MV	12.5	12.5	12.5	MV
7.5	7.5	7.5	7.5	7.5
2.5	2.5	2.5	2.5	2.5
-2.5	-2.5	-2.5	MV	-2.5

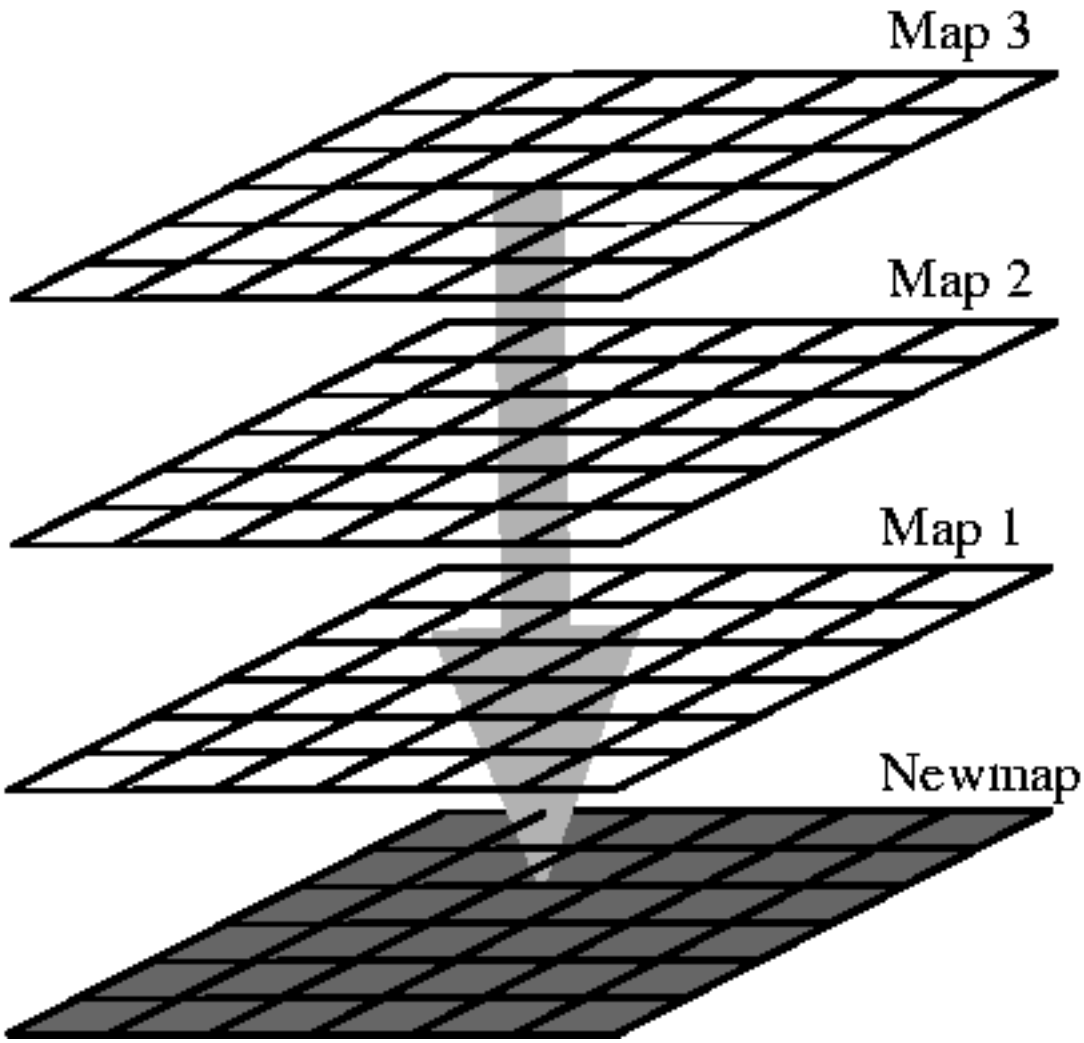
`Expr.map`

MV	MV	0	0	0
MV	1	1	1	0
1	1	1	1	1
1	1	1	1	1
1	1	1	0	1

4.2. General approach to Cartographic Modelling

In PCRaster, a stack of map layers represents the landscape. Each map layer Map1, Map2, Map3,... can be thought of as defining an attribute, containing geographical data for that attribute. The property of each cell is defined as consisting of the set of attribute values stored to that cell on the map layers.

Figure 4-1. Generation of a NewMap as a result of maplayers Map1, Map2, Map3,..



When performing one PCRaster operation from the PCRaster command line the property of each cell is changed by generating a new map layer (representing a new attribute) as a function of one or more existing overlays, see Figure 4-1. So, for each cell the value of the Newmap layer can be expressed as (in conceptual notation, not in PCRaster notation):

Newmap = *f*(Map1,Map2,...)

where Map1,Map2,... are the cell values of one or more overlays in the database and *f* is one of the functions from the set of PCRaster operations. An example of an operation may be the amount of water that accumulates in each cell (Newmap cell values) calculated as a result of an amount of rain on Map1, transported to the cells over the drainage network on Map2.

Instead of using a single function, a PCRaster script for Cartographic Modelling changes the property of a cell according to an instruction given by a set of functions *f*1,*f*2,*f*3,... that use both Map layers already present and the Newmap layers generated during execution of the script. The result may be the creation of several Newmaps containing new values for each cell (this is a conceptual notation, **not** a PCRaster command line):

Newmap1,Newmap2,... =
*f*1,*f*2,*f*3,...(Map1,Map2,...Newmap1,Newmap2,...)

where Newmap1,Newmap2,... are map layers generated during execution of the script. An example may be an extension of the process described above: the amount of water accumulated (Newmap1, created on basis of Map1 and Map2) may result in evaporation (Newmap2) which is a function of both the amount of water accumulated (Newmap1) and landuse (Map3).

In this manual, the PCRaster operators for the functions *f* have been grouped according to the sort of spatial relations that are included in the function. The classes of point operations, neighbourhood operations, area operations and map operations are described in Section 4.3 to Section 4.6.

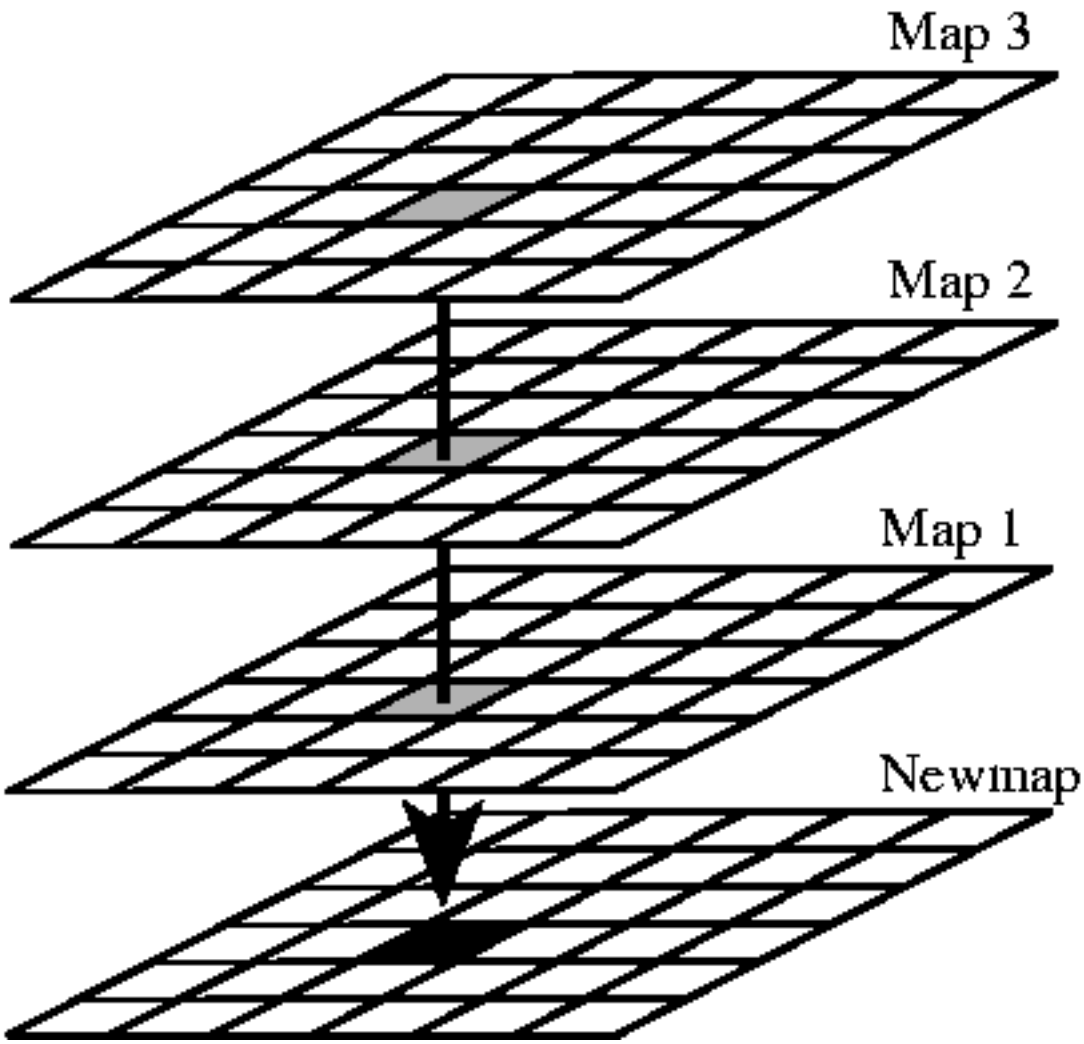
Prev Cartographic Modelling | Home Up | Next Point operations

4.3. Point operations

4.3.1. Introduction to point operations

The class of point operations includes functions that operate only on the values of the map layers relating to each cell (Figure 4-2). The property of a cell is changed on basis of the relations between attributes or the vertical flow of material within the cells: the operation is independent of the property of neighbouring cells (i.e. no relations in lateral direction). In other words, for each cell a new value (stored to a new layer) is calculated on basis of the values in that cell on one or more map layers.

Figure 4-2. Point operation. A new map is generated on a cell-by-cell basis. No lateral relations between cells are included.



4.3.2. Operators for point operations

The simplest of the point operations are the *arithmetic, trigonometric, exponential and logarithmic functions* for mathematical operations such as taking the exponent or sine of the values of one map layer or multiplying cell values of two map layers. Just as simple are operators for *rounding* finding extremes (*minimize, maximize*) or **order**, *comparison operators* and *conditional statements*. For applying Boolean logic *Boolean operators* can be used. Point operations with user specified keys in tables defining relations between map layers can be performed with operators for *relations in lookuptables*. Also random cell values can be generated using *field generation* operators.

Prev: General approach to Cartographic Modelling | Home | Up | Next: Neighbourhood operations

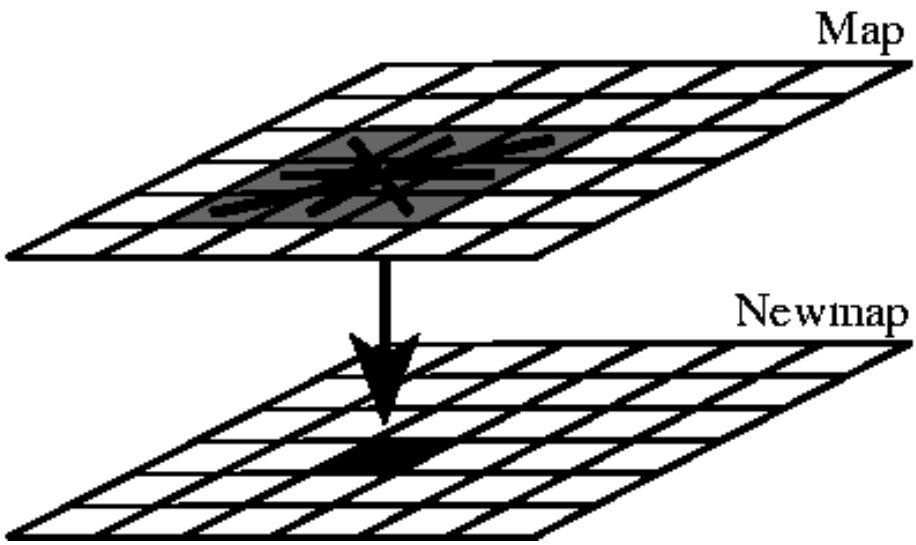
4.4. Neighbourhood operations

4.4.1. Introduction

Neighbourhood operations relate the cell to its neighbours. The property of each cell is changed on basis of some kind of relation with neighbouring cells or flow of material from neighbouring cells. In other words, for each cell a new value is calculated (and stored as a new layer) on the basis of the map layer values in cells that have some kind of spatial association with the cell.

Five categories of spatial association may be represented by the neighbourhood operations. First, the new value of the cell may be calculated on basis of the properties of cells within a specified square window around the cell (Figure 4-3). These so called window operations are described in Section 4.4.2. With the gstat module, it is possible to calculate properties of cells that are in a circular window around the cell.

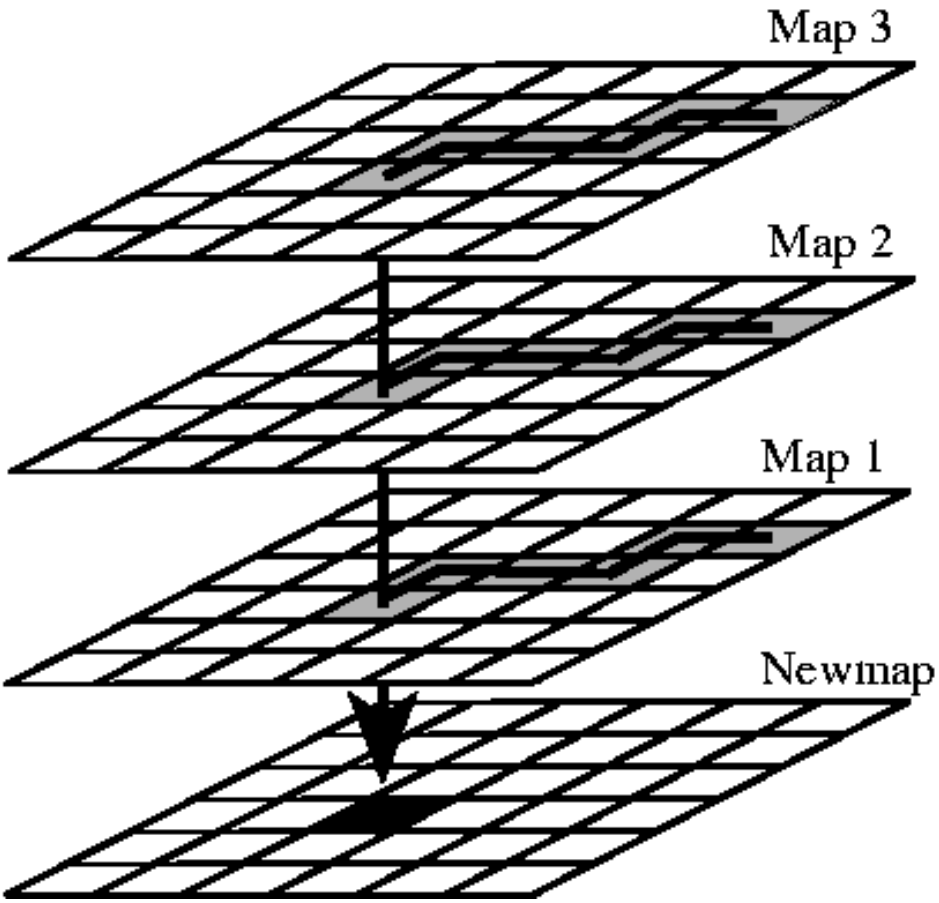
Figure 4-3. Neighbourhood operations within a window.



Second, the new value of the cell may represent the local drain direction to a neighbouring cell in a local drain direction network over a digital elevation model. These are described in Section 4.4.3.

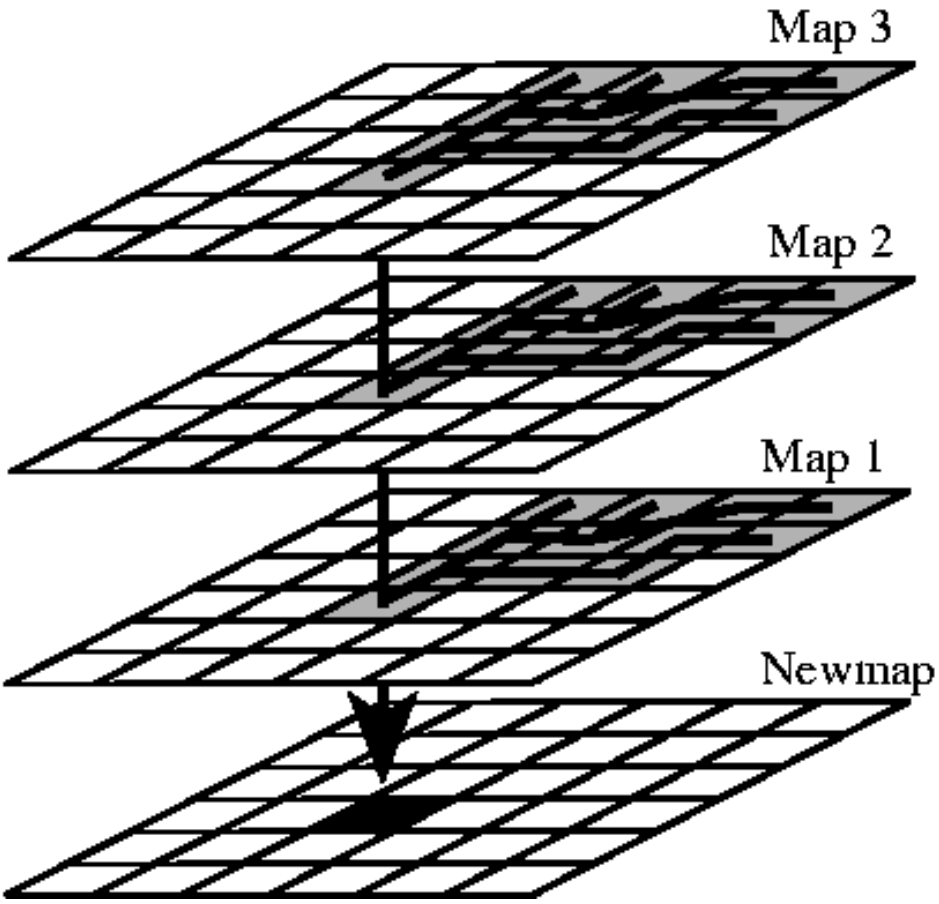
Third, the new value of the cell may be calculated on the basis of the cells that are on a path starting at a given source cell through consecutive neighbouring cells to the cell in question, see Figure 4-4. These operations with friction paths are described in Section 4.4.4. The path represents the shortest distance from the source cell, incorporating friction. Also simply the real distance of the path (for instance the shortest distance to a cell with a garden-restaurant) can be calculated, by specifying a friction of one.

Figure 4-4. Neighbourhood operations over a path. For each cell the NewMap value is calculated on the basis of Map1, Map2, Map3,... values on a path from a source cell.



Fourth, the new value of the cell may be calculated on the basis of cells that are upstream from the cell (i.e in the catchment of the cell, see Figure 4-5). All these operations use a local drain direction map for hydrologic modelling of transport and accumulation of material in a catchment. These are discussed in Section 4.4.5.

Figure 4-5. Neighbourhood operations within the catchment of a cell. For each cell the NewMap value is calculated on the basis of Map1, Map2, Map3,... values in the catchment of the cell, defined by the local drain direction network.



Fifth, the spatial association may be related to the visibility of cells from a target cell, in an elevation model. These neighbourhood operations for visibility analysis are discussed in Section 4.4.6.

4.4.2. Window operations

In a neighbourhood operation within a window, a new value is calculated for each cell on the basis of the cell values within a square window, where the cell under consideration is in the centre of the window.

One can discriminate between two groups of window operations: first each cell value that is calculated may represent a statistical value of the cell values in the window (for instance mean, diversity or extreme values). These operations can also be used to find edges between polygons on a classified map (**windowdiversity** operator). For these operations the size of the square window can be specified by the user and is not restricted to whole magnitudes of cells.

Second square windows of 3 x 3 cells are used for the calculation of land surface topography, when the PCRaster map is a digital elevation model. These operations include the calculation of slope, aspect and curvature within the window.

4.4.3. Local drain direction operations

A local drain direction network is made with the operator **lddcreate** on basis of a map with elevation values.

4.4.4. Friction paths

4.4.4.1. Introduction

This section explains the use of neighbourhood operations for calculating a new value for each cell in the basis of friction cell values on a path between a source cell and the cell under consideration. The accumulation of friction is computed while following a path from a source cell to each cell over a map with frictions (for instance costs). The path that is followed may be determined in two ways. First the paths chosen yield the smallest accumulation of friction. Second the paths may be restricted by the local drain directions on a local drain direction network. In the latter case, the accumulation of friction is calculated for paths in upstream or downstream directions over a local drain direction network.

The operations with friction paths are also used for calculating ordinary real distances over paths (for instance distance to a cell with a railway station). This is done by simply specifying a friction of one.

4.4.4.2. Operations with friction paths

The principle of accumulation of friction is easily explained by an example of a car driving on an asphalt road for 50 km from point A to point B. This will require x litres of fuel. More fuel is needed if the car travels over a rough surface, such as a sandy track, because the friction will be larger. In general the amount of fuel used is equal to the friction distance, which is the distance to travel (*distance*) times the amount of fuel used per distance (*friction*). In this example, the friction depends on the sort of road on which the car has to travel. In an equation:

fuel used = *distance*
x *friction*

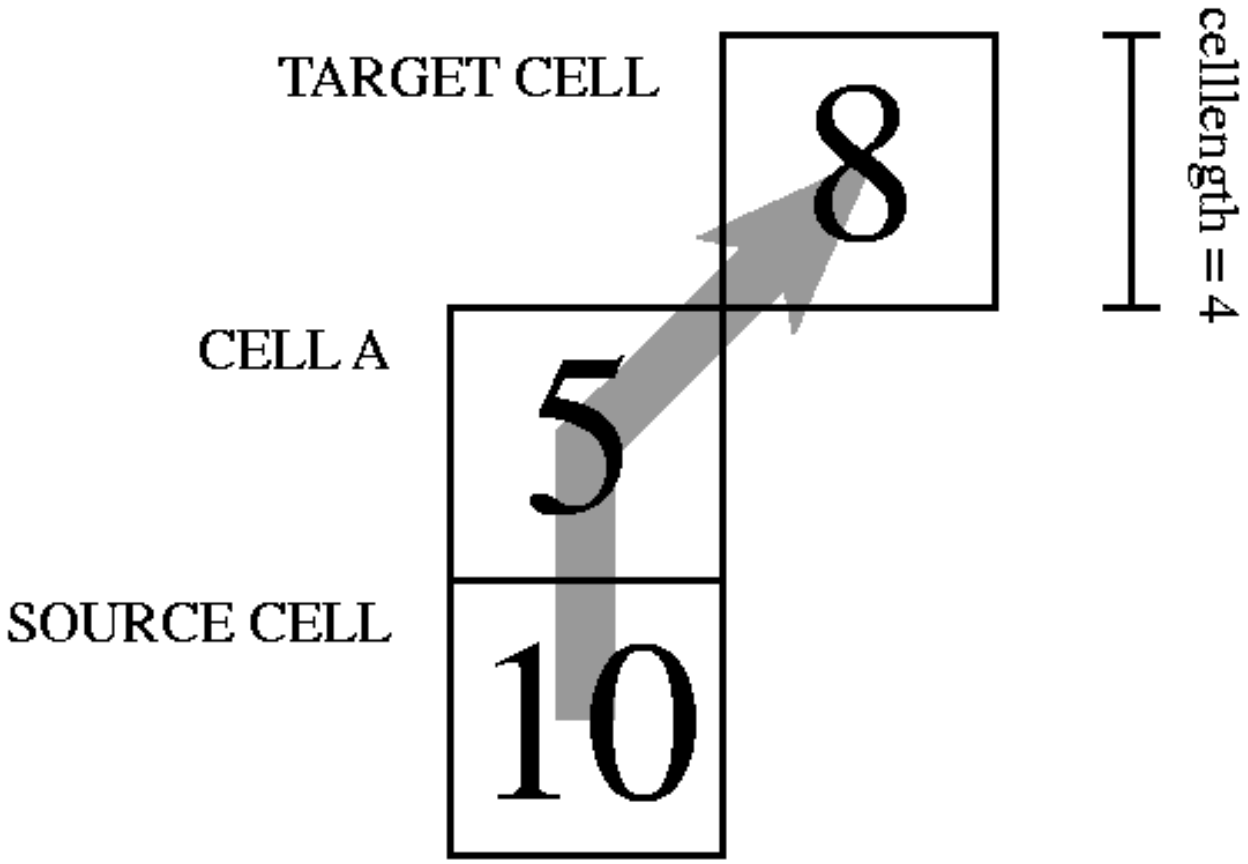
where the fuel used can be regarded as a synonym for the friction distance between point A and B.

All operations with friction use the concept of friction illustrated with the car example. A map with friction cell values is always used to calculate the friction distance between source cells and target cells. The friction values represent some kind of friction per distance unit on the map and may be different between the cells. For instance, the friction may represent the amount of fuel used per unit distance in each cell or the costs of constructing a road per unit distance in each cell. The friction distance ('amount of fuel used') between a source cell and a target cell is calculated by travelling over the path between the cells through consecutive neighbouring cells and calculating the total accumulation of friction distance. Each time that is travelled from one cell to the next the friction distance increases as follows: let *friction*(sourcecell) and *friction*(destinationcell) be the friction values at the cell where is moved from and where is moved to, respectively. While moving from the source cell to the destination cell the increase of friction distance is:

distance
x ((*friction*(sourcecell) +
friction(destinationcell))/2)

where *distance* is the distance between the source cell and the destination cell. Figure 4-6 gives a simple example of a path between a source cell and a target cell travelling through cell A, with a cell length of 4.

Figure 4-6. Path from a source cell to a target cell, crossing cell A.



Let the initial friction distance at the source cell be zero. While moving from the source cell to cell A the friction distance increases by:

{{4 times lpar 10 + 5 rpar} over 2} = 30

,

and when moving from cell A to the target cell in diagonal direction, the friction distance increases by:

{{sqrt{4 sup 2 + 4 sup 2} times lpar 5 + 8 rpar
} over 2} = 36.77

So, the total friction distance between the source cell and the target cell is:

30 + 36.77 = 66.77 .

Of course, on a real map, many possible paths can be found between a source cell and a target cell, through different sets of neighbouring cells. The path which is followed for each target cell can be determined in two ways. The **spread** operation (and the closely related **spreadzone** operation) uses the path that results in the shortest friction distance between a source cell and the cell under consideration. For instance, if you use friction values of 1 the real distance to the source cell is calculated which is nearest as the crow flies (in a straight line). Also the path that is followed for each target cell may be determined by a local drain direction map: the operators **ldddist** and **spreadldd** (and the closely related **spreadlddzone**) calculate the friction distance in upstream and downstream direction from the source cells, respectively. The operator **slopelength** calculates the friction distance in downstream direction from the catchment divide. The above mentioned operations belong to both the group of spread operations and the group of operations with local drain direction maps (with friction like in **spread**) .

4.4.5. Transport of material over a ldd

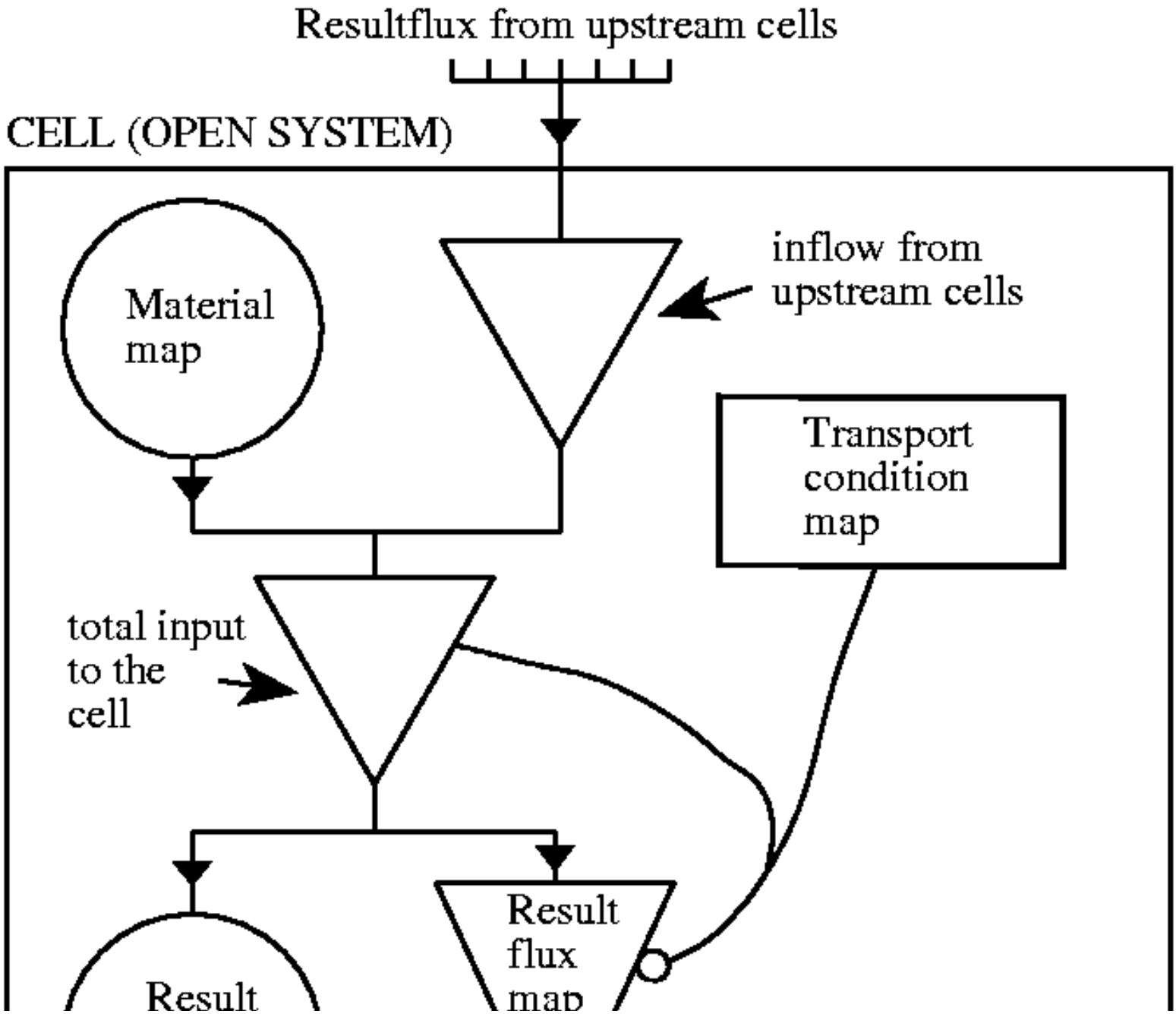
4.4.5.1. Introduction

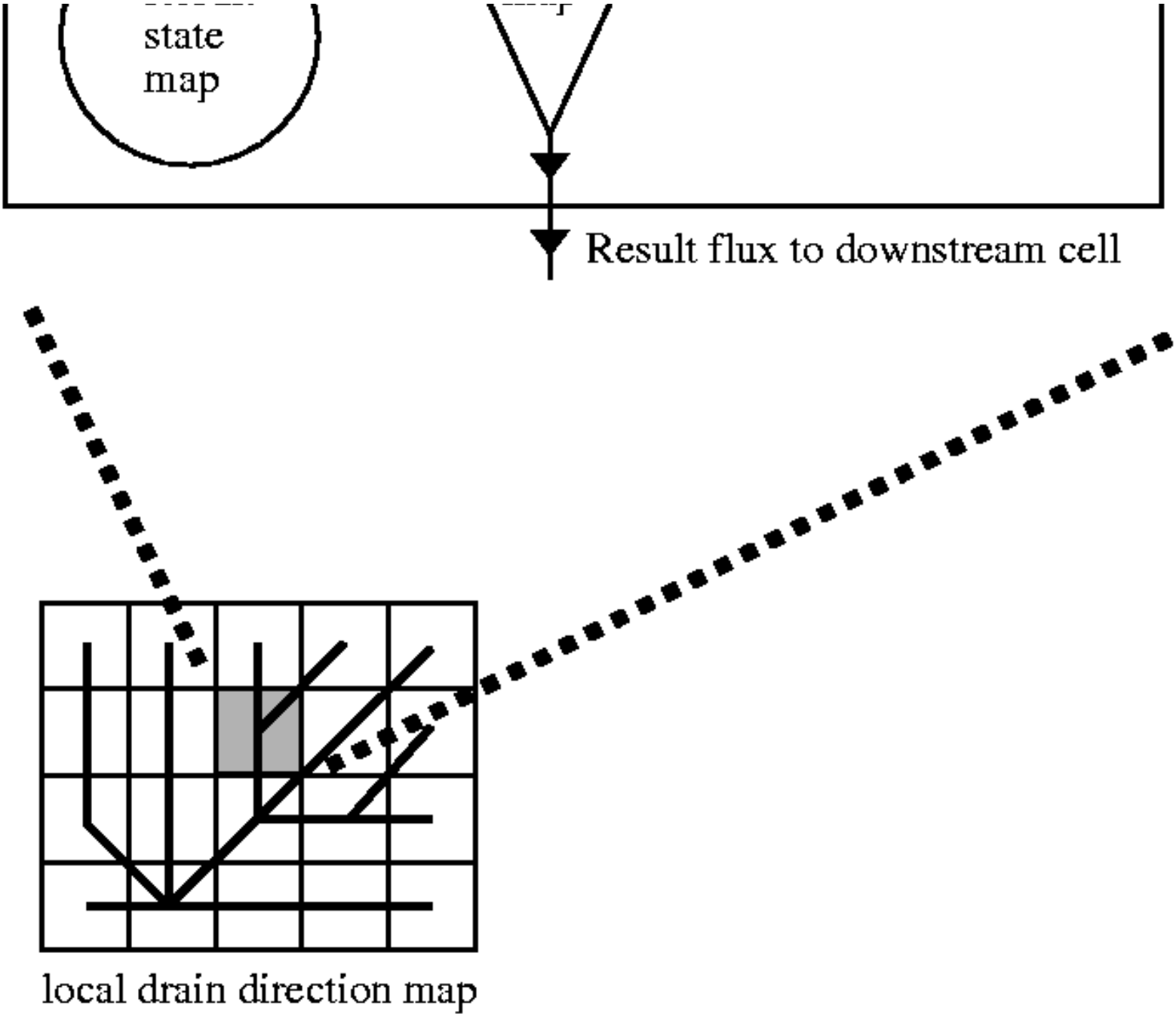
The third group of neighbourhood operations are the operations for hydrologic modelling of transport of material over a local drain direction network. These operations, discussed in the next section, calculate the amount of material transported from upstream cells which is stored in the cell or transported out of the cell.

4.4.5.2. Operations for transport of material over a ldd

The operations for transport of material over a local drain direction network , can be used for modelling processes that include transport in a downslope direction. In most cases these will be used for hydrologic modelling of water transport or for modelling material transported by water.

Figure 4-7. Material transport over a local drain direction network. Bottom: Local drain direction map defining pattern of transport through neighbouring cells. Top: system diagram of an open system representing one cell.





The principle of material transport over this drainage network can be explained using the general systems approach. In this approach each cell is an open system. The direction and pattern of transport of material through the map representing a set of systems is given by the *local drain direction map*: for each cell it defines its upstream neighbours from which material is transported to the cell and its downstream neighbour to which material is transported, see Figure 4-7. The state of the cells at the start of the operation is defined by an input map that gives the amount of material that is available for transport. This *material map* contains for each cell the input of material to the cell at the start of the operation. For instance it may have cell values that represent the amount of rain falling in a cell or the amount of loose soil material that is available in a cell for transport.

The most simple transport operation **accuflux** does not include conditions that impose restrictions on the amount of material that is transported: all material that flows into a cell flows out of the cell. This can be compared with the transport of water over an asphalt landscape, without infiltration or transpiration. But, in most cases a part of the inflow to the cell will be stored or lost in the cell and only the remaining material will be transported out of the cell. A well known example for such a transport process is the transport of water over an unsaturated soil: only the surplus of the amount of water used for saturation of the soil is transported. Several functions can be used to define this division in transport and storage. These are implemented in the different accu... operations. These operations use a *transport condition map* , which for each

cell contains a value related to the transport function. In the example given above it may contain the amount of water needed to saturate the soil in a cell.

Now we describe what happens to the cells on the map during transport (see also Figure 4-7). Transport starts at the cells at the divide of the catchment. For each cell somewhere on the map the total input of material consists of the fluxes of material from upstream cells plus the amount of material at the start of the operation in the cell itself (i.e. the value on the *material map*). This total input is available for transport. A part of the material is stored in the map and is saved as a (new) *result state map* layer, the remaining material flows out of the map, to the downstream cell, and is stored as a (new) *result flux map* layer. The decision about the amount of material that is stored and transported respectively is defined by the sort of accumulation operation that is performed. It is always made on the basis of the cell value of the *transport condition map* with respect to the total input to the cell.

4.4.6. Neighbourhood operations: visibility analysis

The neighbourhood operation for visibility analysis (**view** use a map with elevation cell values over which the cells having direct line of sight from a given viewpoint or viewpoints are determined.

Prev Point operations | Home | Up | Next Area operations

mapattr

Name

mapattr -- Create a new PCRaster map, change or display location attributes of existing PCRaster map

Synopsis

mapattr [options] PCRmap1 PCRmap2....PCRmapn

PCRmap1-n boolean, nominal, ordinal, scalar, directional, ldd spatial

Options

Options that apply to all modes of operations

Set default attribute values with existing map

> --clone *PCRclone* is taken as clone map. This option can be set in the command line or as a global option. If a map attribute is not specified by the user (in the menu or with options) the attribute value of *PCRclone* is assigned to the map that is generated. If *clone* is not set the following defaults are used: data type: boolean; cell representation: small integer; projection: y increases from top to bottom, upper left corner (x_{UL}, y_{UL}): (0,0), cell length 1 and angle 0.

Print information for all maps

-p

> gives information for all maps PCRmap1, PCRmap2,...PCRmapn (from top to bottom): number of rows, number of columns, cell length, data type, cell representation, projection, angle in degrees, upper left corner (x_{UL}, y_{UL}), minimum value, maximum value, PCRaster version number, file_id (i.e. the number of the file, used internally by PCRaster), native/not native (specifies whether the byte order of the file is native ('y' is printed) or not native ('n' is printed): big endian or little endian), attribute table (specifies whether the file contains additional attributes ('y' is printed) like for example a legend or a colour palette or not ('n').

Options that apply to operation with the menu

Change location attributes of an existing map

-e

starts the menu allowing to change the location attributes projection, upper left corner (x_{UL}, y_{UL}), cell length and angle of the existing map PCRmap1. The other map attributes are fixed. Only one map (PCRmap1) can be specified. Do not use -e in combination with other options.

Copy location attributes of a map to other maps

-c

when using this option two or more PCRmap1, PCRmap2,...PCRmapn must be specified. These maps must have corresponding number of rows and corresponding number of columns. The location attributes cell length, projection, angle, upper left corner (x_{UL}, y_{UL}), of the first map PCRmap1 are copied to the other map(s) PCRmap2,...PCRmapn. The other map attributes are not changed. Do not use -c in combination with other options.

Options that result in operation without the menu
If one of the following options are set, **mapattr** does not invoke the menu. So all attributes must be defined with the following options else the default values (or the values of *PCRclone*, see the global cloneoption) are assigned.

-R *NumberOfRows*

Specifies the number of rows in the map that is generated. *NumberOfRows* must be a whole positive value.

-C *NumberOfColumns*

Specifies the number of columns in the map that is generated. *NumberOfColumns* must be a whole positive value.

-B,-N,-O,-S,-D or -L

Specifies the data type of the map that is generated (respectively boolean, nominal, ordinal, scalar, directional, ldd). Default: boolean, or if --clone is set, the data type of the clone map PCRclone.

--single or --double and --small or --large

In most case, the default cell representation will be sufficient. If you want, you can specify the cell representations:
Scalar and directional data types

--single

cell values are represented by single real cell representations (default, single, precision)

--double

cell values are represented by double real cell presentations (double precision)

Nominal and ordinal data types

--small

cell values are represented by small integer cell representaiton (default)

--large

cell values are represented by large integer cell representation

-P yt2b or -P yb2t

Specifies the projection: -P yt2b, y increases from top to bottom or -P yb2t, y increases from bottom to top. Default: yt2b or if --clone is set the projection of *PCRclone*.

-x *XCorULC*

Specifies the x-coordinate of the upper left corner of the map that is generated. *XCorULC* is a real value. Default: 0.0 or if --clone is set, the value of the clone map *PCRclone*.

-y *YCorULC*

Specifies the y-coordinate of the upper left corner of the map that is generated. *YCorULC* is a real value. Default: 0.0 or if --clone is set, the value of the clone map *PCRclone*.

-l *CellLength*

Specifies the celllength of the map that is generated. *CellLength* is a real value. Default: 1 or if --clone is set, the value of the clone map *PCRclone*.

-a *Angle*

Specifies the angle of the map that is generated. *Angle* is the angle in degrees between -90 and 90 degrees. Default: 0.0 or if --clone is set, the value of the clone map *PCRclone*.

Operation

The **mapattr** operator generates a new PCRaster map with map attributes specified by the user, changes the location attributes of an existing map or prints map attribute information.

Operation with the menu
A new map is generated by specifying one input file `PCRmap1` and not setting options. The operator invokes a menu. In the menu the location attributes can be entered of the new map `PCRmap1` that is created. You can scroll through the menu with the arrow up or arrow down keys or the keys listed at the bottom of the menu. An menu item is entered by pressing <Enter> and typing the value followed by <Enter>. The menu items 'data type' and 'projection' are filled in by selecting one of the options with the arrow left or arrow right keys (or one of the keys listed at the bottom of the menu) instead of typing the entry. Quit the menu by pressing 'q'. The program asks you whether the map must be created. You can answer by pressing 'Y' (Yes, create the map), 'N' (No, do not create the map and leave the menu) or by pressing <Escape which effects that you can resume editing.

Location attributes of one map can be changed by specifying one existing map `PCRmap1` and the option -e. Note that this is not meant for cutting or resampling the map which is done with the operator **resample**. See also the section on the import map type.

Location attributes of the first, existing, `PCRmap1` are copied to the existing `PCRmap2`,...`PCRmapn` with

the option -c.

Map attribute information of multiple PCRmap1-n is printed with the option -p.

Operation without the menu
The map attributes can also be entered without the menu, by setting options. If one of the options -R, -C, -B,-N,-O,-S,-D,-L, -P, -x, -y, -l and/or -a for specifying the map attributes from the command line are set, the operation is performed without the menu. All modes of operation possible with the menu can also be executed in this command line/no menu mode. So you can also use the options -e or -c for changing map attributes of an existing map and copying map attributes from one map to other maps respectively.

If one of the map attributes is not specified with an option in the command line, **mapattr** assigns the default value or, if a clone map is specified with the option --clone, the value of the *PCRclone*. See also the --clone option in the option list at the top of the **mapattr** description.

See Also

resample

Examples

Generation of a new map that is stored under the filename 'mask.map'; invokes the menu:

```
mapattr mask.map
```

Changing the location attributes of an existing map *clone.map*; invokes the menu:

```
mapattr -e clone.map
```

Copying the location attributes of `clone1.map` to `dem.map` and `ldd.map`; does not invoke the menu:

```
mapattr -c clone1.map dem.map ldd.map
```

Printing (on the screen) the map info for `clone1.map`, `dem.map` and `ldd.map`:

```
mapattr -p clone1.map dem.map ldd.map
```

Generation of a new map that is stored under the filename `mask2.map`; does not invoke the menu (here, the option -a is not set so the angle of the map is assigned the default value 0):

```
mapattr -R 19 -C 68 -B -P yb2t -x 12 -y -14.02 -l 0.8 mask2.map
```

Prev Home Next
map2col Up resample

table

Name

table -- Creates on basis of one or more maps a table with a score for each key in the table.

Synopsis

`table [options] PCRmap1 PCRmap2.....PCRmapnResulttable`

`PCRmap1-n`	boolean, nominal, ordinal, scalar, directional, ldd spatial; they must have the same location attribute
`Resulttable`	ascii table

Options

--unittrue or --unitcell

--unittrue

the score is the total true area of the cells that match a key (default)

--unitcell

the score is the number of cells that match a key

--degrees or --radians

--degrees

directions of directional data type are in degrees (default)

--radians

directions of directional data type are in radians

--columntable or --matrixtable}

--columntable

--Resulttable will be a column table (column table format).

--matrixtable

if two maps --PCRmap1, --PCRmap2 are specified in the command line `Resulttable` will be a matrix in matrix table format, instead of a table. The options -i and -m are ignored.

keys defined automatically

-n *nrintervals*

the number of intervals in the key column for scalar and directional data type (default 8)

-h

the intervals in the key column for scalar and directional data type are histogram stretched: intervals are chosen that result in an (almost) equal number of cells into each interval.

-H *nrhistslots*

corresponds with -h, additionally the number of histogramslots (*nrhistslots*) can be given. With the default value 1024 the number of intervals in each interval will be almost the same, but not exactly. Choosing a larger *nrhistslots* will improve the accuracy the intervals are chosen, on the contrary it will take longer to determine the intervals.}

keys defined by the user: -i

-i *Inputtable*

the *Inputtable* is an ascii table that defines the keys used for calculation of the scores. For explanation, see Operation.

remove keys with score zero

-0

all keys, including these with a zero score, are printed in `Resulttable`. Default: keys with a zero score are not printed.

change order of columns mode, for column tables only

-m *columnnrmove*

Use in combination with -i *Inputtable* and do not specify input maps `PCRmap`1, `PCRmap`2,...`PCRmap`n. In *Inputtable* the column with column number *columnnrmove* is moved to the last column. The table with changed column order is saved as `Resulttable`.

Operation

Operation with a column table (with global option)

Example 1. Example of a column table generated with the table operation with two input maps and the setting -n 4. The first and second column give the values of `PCRmap`1 (data type scalar) and `PCRmap`2 (data type nominal) respectively; the third column contains the score fields.

<0,2] 1 205
<2,4] 1 123
<4,6] 1 142
<6,8] 1 0
<0,2] 2 10
<2,4] 2 350
<4,6] 2 4209
<6,8] 2 2

The maps `PCRmap`1, `PCRmap`2,...`PCRmap`n are PCRaster maps with the same location attributes. In the `Resulttable` relations between the cell values on these maps are given. Combinations of cellvalues on `PCRmap`1, `PCRmap`2,...`PCRmap`n can be specified in keys or are determined automatically. For each combination

(key) the number of cells that matches the combination is counted and added to the key in the Resulttable.

The Resulttable will consist of a number of *n*+1 columns. The first *n* columns are key columns, where *n* is the number of the one or more maps PCRmap1, PCRmap2,...PCRmapn. The key columns consist of key fields; each key field is one value or a range of values. The key fields in the first column have been linked to cell values on PCRmap1, the key fields in the second column to values on PCRmap2, and so on, where the key fields in the *n*th column have been linked to values on PCRmapn. The last column (column number n+1) contains so called score fields with the scores. Each row in the Resulttable is called a tuple. Of course, a row consists of *n* key fields and one score field.

For a tuple in the Resulttable, the **table** operator has counted the number of cells that match the key fields in the tuple. This number multiplied with the area of one cell (if the option --unitcell is set: the number of cells) has been assigned to the score field in the tuple. So for each cell, the value on PCRmapi (where *i* is 1 to *n*) has been compared with the *i*th key field of the tuple. The PCRmapi value of the cell matches the key field if it is equal to the value in the key field or if it is in the range of the key field, in case of a key field consisting of a range of values. If all cell values on the maps match the key fields belonging to them the cell matches the tuple and has been included in the score. This counting has been done for each tuple.

The Resulttable is an ordinary ascii text file. You can display the table by typing the DOS command 'type' followed by a space and the table name Resulttable. It has the following lay-out. In a row (tuple), the number of key fields equals the number of maps PCRmap1, PCRmap2,...PCRmapn. The key fields are followed by the score field. The fields are separated by one or more spaces or tabs. A key field is a single value, or a range of values, where a range of values is typed as: '[' or '<' symbol, minimum value, comma, maximum value, ']' or '>' symbol. The minimum and maximum values are included in the range if square brackets (respectively '[' and ']') are used, they are not included if '<' or '>' are used. A value which is omitted in the range definition means infinity. Examples of tuples are (assume a cellarea of 100 m_2):

one PCRmap1, PCRmap2,...PCRmapn.:

[,0.05> 37600

An area of 37600 (376 cells of 100 m_2) has PCRmap1 values smaller than 0.05.

#two PCRmap1, PCRmap2,...PCRmapn:

[-1.42,-0.2> [,9> 20800

An area of 20800 (208 cells of 100 m_2) has an PCRmap1 value equal to -1.42 or between -1.42 and -0.2 and an PCRmap2 value smaller than 9.

Default the keys are determined automatically, before the counting of cells starts. The way this is done depends on the data type of the maps PCRmap1, PCRmap2,...PCRmapn:

boolean, nominal, ordinal, ldd data type

the key column consists of sets of all whole values between and including the maximum and minimum cell value on the map linked to the key column.

scalar data type

the key column consists of sets of ranges; default 8 ranges of equal width, or specified by the options -n *nrintervals*; -h or -H *nrhistslots*.

directional data type

the key column consists of sets of: the number -1 for the value -1 (no direction) and *nrintervals*-1 ranges for the directions, the number of ranges used can be specified by -n *nrintervals*; -h or -H *nrhistslots*; default, 7 ranges of equal width are used.

Example 2. Example of a matrix table generated with the table operation with the settings --matrixtable and -n 4. Same input maps were used as in the first table (shown above). The fields in the first row contain ranges of PCRmap1; the fields in the first column contain values of PCRmap2. The field in the top left corner is a dummy field. The remaining fields are score fields.

999 <0,2] <2,4] <4,6] <6,8]
1 205 123 142 0
2 10 350 4209 2

The keys can also be given by the user, specifying an *Inputtable* with the option -i. The *Inputtable* must have the same lay-out as the Resulttable described above with the exception that the column with the scores does not need to be given (of course it is generated by the **table** command). The *Inputtable* can be made using your favourite text editor program or with a spread sheet or word processing program (export as text file!). If you want to change the order of the columns in a table you have made, use **table** with the option -m. A Resulttable created with **table** can also be used as *Inputtable*.

The key columns of *Inputtable* will be linked to the input PCRaster maps: the first column to PCRmap1, the second to PCRmap2,..., the *n*th to PCRmapn. If *Inputtable* contains more columns than the number of *n* input maps specified in the command line, the (*n* + 1)th columns and further are deleted by **table** before execution of the operation. The scores are written to the (*n* + 1) column.

Each key column must contain key values in the domain of the data type of the map which is linked to the key:

boolean, nominal, ordinal, ldd data type

whole values

scalar data type

values or ranges;

directional data type

the value -1; or values or ranges in the domain of the data type [0,360> (option --degrees), [0,2pi> (option --radians).

operation with a matrix table (with the option --matrixtable)
With the option --matrixtable set, with two maps PCRmap1, PCRmap2 specified in the command line, Resulttable will be a matrix table instead of a column table. If the option --matrixtable is set and a different number of PCRmap1, PCRmap2,...PCRmapn is specified, a column table is generated and the operation is performed as described above.

The matrix will have the following lay-out. The first field in the top left corner has no meaning, it is a dummy field. The first row consists of this dummy field and the key fields which have been linked to PCRmap1. The first column consists of the dummy field and the key fields which have been linked to PCRmap2. The remaining fields in the matrix are score fields. Each score field contains the number of cells that have an PCRmap1 value of the key field of its column and have an PCRmap2 value of the key field of its row.

If in addition to --matrixtable, the option -i *Inputtable* is used, the input table must have the following lay-out. The first row of *Inputtable* consists of a dummy field and the key fields which will be linked to PCRmap1. The first column consists of the dummy field and the key fields which will be linked to PCRmap2. The remaining fields

must be filled in with arbitrary values; these will be replaced by the scores.

Group

This operation belongs to the group of Point operators; relations in tables

See Also

lookup

Examples

1.
```
table -n 4 PCRmap1.map Result1.txt
```
PCRMap1.map

10	12	4	14	12
1	19	0	21.5	-85.5
10	12	7.5	6	15.1
8	13	1	19	4
16	10	11	9	8

Result1.txt
```
[-85.5 ,-58.75> 1
[-58.75,-32   > 0
[-32   ,-5.25 > 0
[-5.25 ,21.5  ] 24
```

2.
```
table -n 4 -h PCRmap1.map Result2.txt
```
PCRMap1.map

10	12	4	14	12
1	19	0	21.5	-85.5
10	12	7.5	6	15.1
8	13	1	19	4
16	10	11	9	8

Result2.txt
```
[-85.5  ,6.03516> 7
[6.03516,10.0059> 7
[10.0059,13.0361> 5
[13.0361,21.5   ] 6
```

3.
```
table -i Input.txt PCRmap1.map PCRmap2.map Result3.txt
```

PCRMap1.map

10	12	4	14	12
1	19	0	21.5	-85.5
10	12	7.5	6	15.1
8	13	1	19	4
16	10	11	9	8

Result2.map

2	2	2	2	2
2	2	2	2	3
2	2	2	2	3
2	2	3	3	3
3	3	3	3	3

Input.txt

```
<,6] 2
<6,> 2
<,6] 3
<6,> 3
```

Result3.txt

```
< ,6] 2 4
<6, > 2 11
< ,6] 3 3
<6, > 3 7
```

4.

```
table -m 2 -i Input2.txt Result4.txt
```

```
Input2.txt    Result4.txt
2 <1,5] 2 0   2 2 0 <1 ,5 ]
2 <5,10] 1 1  2 1 1 <5 ,10]
2 <10,20] 3 1 2 3 1 <10,20]
3 <20,30] 4 1 3 4 1 <20,30]
```

Prev resample | Home Up | Next timeplot

II. PCRaster applications

Table of Contents

asc2map -- Converts from ascii file format to PCRaster map format

col2map -- Converts from column file format to PCRaster map format

legend -- Attaches a legend to or changes the legend of one or more maps.

map2asc -- Converts from PCRaster maps format to ascii file format

map2col -- Converts from PCRaster map format to column file format

mapattr -- Create a new PCRaster map, change or display location attributes of existing PCRaster map

resample -- Cuts one map or joins together several maps by resampling to the cells of the result map.

table -- Creates on basis of one or more maps a table with a score for each key in the table.

timeplot -- Plots timeseries in a x,y lineplot on the computer screen.

asc2map

Name

asc2map -- Converts from ascii file format to PCRaster map format

Synopsis

asc2map [options] asciifile PCRresult

asciifile asciifile

PCRresult specified by data type option; if data type option is not set: data type of *PCRclone* spatial

Options

Options can be given related to the layout of asciifile and the way asciifile must be read. These options are described in the operation section. Other options are:

--clone *PCRclone*

PCRclone is taken as clonemap. If you have set a global clonemap as global option, you don't need to set **clone** in the command line: the clonemap you have set as global option is taken as clonemap. If you have not set a global clonemap or if you want to use a different clonemap than the global clonemap, you must specify the clonemap in the command line with the **clone** option.

-B, -N, -O, -S, -D and -L

This data type option specifies the data type which is assigned to PCRresult (respectively boolean, nominal, ordinal, scalar, directional, ldd). If the option is not set, PCRresult is assigned the data type of *PCRclone* or the global clone. The data in asciifile must be in the domain of the data type which is assigned to PCRresult. For description of these domains see the description of the different data types

--single or --double and --small or --large

In most case, the default cell representation will be sufficient. If you want, you can specify the cell representations:
Scalar and directional data types

--single

cell values are represented by single real cell representations (default, single, precision)

--double

cell values are represented by double real cell presentations (double precision)

Nominal and ordinal data types

--small

cell values are represented by small integer cell representaiton (default)

--large

cell values are represented by large integer cell representation

if option -D is set; --degrees of --radians

--degrees

values on `asciifile` are interpreted as degrees (default)

--radians

values on `asciifile` are interpreted as radians

-m *nodatavalue*

nodatavalue is the value in *columnfile* which is converted to a missing value on `PCRresult`. It can be one ascii character (letters, figures, symbols) or a string of ascii charaters. For instance: -m -99 -98. or -m j83s0w. Default, if this option is not set, 1e31 is recognized as a missing value.

-s *separator*

By default, whitespace (one or more tabs, spaces) is recognized as separator between the values of a row in the `asciifile`. If the values are separated by a different separator, you can specify it with the option. The *separator* can be one of the ascii characters (always one). In that case, **asc2map** recognizes the specified separator with or without whitespace as separator. For instance, if the values in `asciifile` are separated by a ; character followed by 5 spaces, specify -s ; in the command line (you do not need to specify the whitespace characters).

Operation

The *asciifile* is converted to `PCRresult`, which is an expression in PCRaster map format. `PCRresult` is assigned the location attributes of *PCRclone* (number of rows and columns, cell size, x and y coordinates), or if the option --clone is not set in the command line the location attributes of the global clone. The `asciifile` must contain data values separated by one or more spaces or tabs. Values may contain the characters: -eE.0123456789. Valid values are for instance:

-3324.4E-12 for -3324.4×10^{-12}
.22 for 0.22

simple conversion

The most simple conversion is a conversion ignoring the layout of your data on the asciifile (ordering of data by rows, row definitions or headers for instance). This simple conversion is performed default. All the characters on your asciifile will be interpreted as data. The operator scans the asciifile starting at the top line from left to right, than the second line from left to right etc. Each time a value is scanned it is assigned to a cell on PCRresult until PCRresult is totally filled with cell values. If the asciifile contains a larger number of values than the number of cells on PCRresult, the remaining values are simply ignored. The values are assigned to PCRresult starting with the top row on the map and ending with the bottom row. The first value which is filled in is the first value in the asciifile, the second value is the second value in the asciifile etc..
This conversion imposes almost no restriction on the layout of the asciifile: if your data *are* ordered in a number of rows and columns which corresponds with the number of rows and columns on PCRresult it will result in a correct conversion, but if they are not ordered this way (for instance they are on one line in the asciifile) a conversion is also possible.

conversion from ARC/INFO ascii files
In ARC/INFO, grid maps can be converted to an formatted ascii file using the ARC/INFO gridascii command. These output files from ARC/INFO are converted to the PCRaster map format with **asc2map** using the option -a without setting the options -s, -m, -h and -r. These latter options will be totally ignored if you set them in combination with -a. The output asciifile from ARC/INFO will contain a header. The number of rows and columns of the original ARC/INFO map given in the header must correspond with the number of rows and columns of *PCRclone*. The remaining location attributes in the header are ignored during conversion since they are taken from *PCRclone* (cell size and x,y coordinates). If the header contains a no_data_value, each value in the asciifile which corresponds with the no_data_value is assigned a missing value on PCRresult. If the header does not contain a no_data_value the value -9999 is recognized as a missing value.

conversion from Genamap ascii files
In Genamap, grid maps can be converted to an formatted asciifile using the Genamap audit command. These output files from Genamap are converted to the PCRaster map format with **asc2map** using the option -g. The number of rows and columns of the original Genamap map, given in the header of the output file from Genamap must correspond with the number of rows and columns of *PCRclone*. Assignment of missing values can be specified by the option -m. Do not use the options -s, -h and -r in addition to -g. If you do set them, they will be totally ignored.

conversion from asciifiles with an exotic format
Two options can be used to impose the command to take into account the layout of your asciifile. They can not be used in combination with the options -a and -g.
-h *asciilinesheader*
This is used if the asciifile contains a header with information which must be ignored during scanning. The option -h must be followed by *linesheader* which must be whole number larger than 0. This is the number of lines which will be skipped at the top of the asciifile. The asciifile is scanned starting at line *linesheader*.
-r *asciilinesbeforemaprow*
The option -r results in skipping of data in asciifile each time before **asc2map** starts with filling a new row on PCRresult. Rows on PCRresult are filled in as follows:
First a number of lines on asciifile is skipped. The number of lines which is skipped is given by *asciilinesbeforemaprow*, it must be a whole value equal to or larger than 0. Then, the asciifile is scanned

until the first row on `PCRresult` is filled with data. At that point, the remaining data on the line in `asciifile` are skipped plus data on the next *asciilinesbeforemaprow* number of lines. Then, the next row on `PCRresult` is filled with the data read from the row on `asciifile` after the skipped rows.

Notes

Using **asc2map** for generating a `PCRresult` of data type ldd is quite risky: probably it will result in a ldd which is unsound. If you do want to create a `PCRresult` of data type ldd use the operator **lddrepair** afterwards. This operator will modify the ldd in such a way that it will be sound.

Group

This operation belongs to the group of Creation of PCRaster maps

See Also

col2map

Examples

1.
```
asc2map --clone mapclone.map -S -m mv -v 4 AscFile1.txt Result1.map
```

Result1.map

210	2.5	3	8
2.7	-0.5	0	4
MV	3.2	0.01	2

AscFile1.txt

```
210    2.5 3    8
2.7   -0.5 0    4
MV     3.2 0.01 2
```

mapclone.map

1	1	1	1
1	1	1	1
1	1	1	1

2.
```
asc2map --clone mapclone.map -D -a AscFile2.txt Result2.map
```

Result2.map

MV	0	2.3	8.9
0.8	351	351	0
45	MV	10	10

AscFile2.txt

```
NCOLS 4
NROWS 3
XLLCENTER 120
YLLCENTER 120
CELLSIZE 15
NODATA_VALUE -9999
-9999  0      2.3   8.9
 0.8   351   -9    360
45    -9999 370    10
```

mapclone.map

1	1	1	1
1	1	1	1
1	1	1	1

Prev Home Next
PCRaster applications Up col2map

legend

Name

legend -- Attaches a legend to or changes the legend of one or more maps.

Synopsis

legend [options] PCRmap1 PCRmap2.....PCRmapn

PCRmap1-n boolean, nominal, ordinal; they must have the same data type
spatial

Options

By default, **legend** starts a menu. Options that apply to this default menu mode are -l and -h.

enter also labels for cell values with a smaller value than which occurs on PCRmap11....PCRmapn
-l *minvalues*

the menu will allow you to fill in labels for whole cell values equal to or between the (whole) value *minvalue* and the maximum value on the PCRmap1,PCRmap2,...PCRmapn.

enter also labels for cell values with a higher value than which occurs on PCRmap11....PCRmapn
-h *maxvalues*

the menu will allow you to fill in labels for whole cell values equal to or between the minimum value on the PCRmap1,PCRmap2,...PCRmapn and the (whole) number *maxvalue*.

The operator can also be used in a mode without the menu. The options that invoke this mode are described in the second part of the operation section.

Operation

The one or more maps PCRmap1,PCRmap2,...PCRmapn must have the same data type (boolean,nominal or ordinal), the location attributes do not need to correspond. The PCRmap1,PCRmap2,...PCRmapn may have (corresponding or different) legends attached or may not have legends attached. The **legend** operator will overwrite or assign in both cases the same legend to all PCRmap1,PCRmap2,...PCRmapn.

Operation with the menu

This is the default action when no options or the options -l or -h are set. The operator invokes a menu which is used to change or enter the legend labels for the cell values that are found on one or more PCRmap1,PCRmap2,...PCRmapn. In addition the title of the legend can be assigned. The keys that are supported by the menu are given at the bottom of the menu.

Operation without the menu
The operator can also be used without the menu. This is done by setting one of the following options.

copy the legend of the first map to the other maps

> -c: More than one PCRmap1,PCRmap2,...PCRmapn are specified. The legend of the first map PCRmap1 is copied to the other maps PCRmap2,...PCRmapn.

store the legend labels in an ascii formatted legend file

> -w *outputlegendfile*: One or more PCRmap1,PCRmap2,...PCRmapn may be specifed. The cell values of these maps with the labels are stored in the ascii formatted *outputlegendfile*. For each cell value, the label is stored of the PCRmapi (*i* is 1...*n*) with the lowest *i* that contains the cell value under consideration. The layout of the legend file *outputlegendfile* is given below.

read the legend labels for the maps from an ascii formatted legend file

> -f *inputlegendfile*: One or more PCRmap1,PCRmap2,...PCRmapn may be specified. The labels given in the *inputlegendfile* are assigned to the legend of these maps. A cell value on an input map that does not occur in the *inputlegendfile* is assigned a label. The layout of the legend file *inputlegendfile* is given below.

The general layout of a legend file is as follows. The ascii formatted file constists of two columns, separated by one or more whitespace characters (space(s), tab(s)). The cell values are in the first column, the labels are in the second column. The first row contains in the first column the field -0 and in the second column the title of the legend. The following rows contain in the first column the cell value and in the second column the label for that cell value.

When the legend file is the output from the **legend** operator, the title row (if a title occurs in one or more of the input maps) is the first row. The cell values with the labels are written in rising order from top to bottom, starting with row two.

When the legend file is used as input file, the row with the title may be omitted. If it is given it *must* be the first row of the file. The following rows contain the cell values with the labels. The order of these rows does not matter.

An example of an input legend file is:

```
-0  landuse
4  arable land
1  woodland
3  buildings
2  lake
2  river
```

See Also

Prev col2map | Home | Up | Next map2asc

map2asc

Name

map2asc -- Converts from PCRaster maps format to ascii file format

Synopsis

map2asc [options] PCRmap asciifile

PCRmap	boolean, nominal, ordinal, scalar, directional, ldd spatial
asciifile	formatted asciifile

Options

Options can be given for assigning a special layout to asciifile. These options are described in the operation section. Other options are:

-m *nodatavalue*

> *nodatavalue* may be one ascii character (letters, figures, symbols) or a string of ascii charaters. For instance: -m -999, -m ? or -m 8kf.f. The missing values on the map will be assigned the *nondatavalue* on the asciifile

-s *seperator*

> *separator* may be one ascii character (letters, figures, symbols, space, tab) or a string of ascii characters. A space or tab is specified using "\" \"", for instance: -s "\" \"" is a space as separator. The cell values which are on one row on PCRmap will be separated by *separator* on asciifile. Default, if -s is not set, **map2asc** prints one space *before* each cell value field. Note that if you want to convert the asciifile back to PCRaster map format with the **asc2map** operator, it should contain whitespace characters only or whitespace characters with only *one* non whitespace character as separator.

-f C *-typeformat*

> This option is used to specify the sort of format which is assigned to the cell values in asciifile. See the discussion of -f in **map2col**.

Operation

simple conversion
The data in PCRaster map format on `PCRmap` are converted to ascii format and saved as `asciifile`. Default a simple conversion is performed. This is a rowwise conversion to an `asciifile` without header. The rows on `PCRmap` are scanned from left to right, starting with the top row and ending with the bottom row. Each time a value is scanned it is added to `asciifile`, starting with a new line on `asciifile` if a new row on `PCRmap` is scanned. The values in one row on `PCRmap` will be on one line on `asciifile`, with a separator defined by the option - s (default one or more spaces, see above). The lines on `asciifile` always end in a cell value.

conversion to ARC/INFO input format
In ARC/INFO ascii data with a special lay-out containing a header with location attributes and the *nodatavalue* can be imported using the ARC/INFO command asciigrid. An `asciifile` with the lay-out needed for this command can be created with **map2asc** specifying the option -a. Additionally only the option -m can be specified.

conversion to other ascii file lay-outs
Two options can be set to specify `asciifiles` with other lay-outs:

-c

Default a rowwise output is performed: lines on `asciifile` correspond with rows on `PCRmap`. If this option is set the output will be columnwise: the first column on `PCRmap` (from top to bottom) is printed as the first line (from left to right) on `asciifile`, the second column as the second line etc. The number of columns on `PCRmap` will correspond with the number of rows on `asciifile`.

-n

numberofcellsonline must be a whole value larger or equal to 1. Default, each row of cell values on `PCRmap` is saved as one line in `asciifile`. This option can be used to print a different number of cell values (the value of *numberofcellsonline*) on one line in `asciifile`.

Notes

Group

This operation belongs to the group of Data export from PCRaster map

See Also

map2col

Examples

1.

```
map2asc -m -m PCRmap.map AscFile1.txt
```

AscFile1.txt

```
      123.23          m          0
        45.5        124      0.419
       56.34         32      0.016
```

PCRmap.map

123	MV	0
45.5	124	0.419
56.3	32	0.016

2.

```
map2asc -m m -s s -f 3.1f PCRmap.map AscFile2.txt
```

AscFile2.txt

```
123.2sms0.0
45.5s124.0s0.4
56.3s32.0s0.0
```

PCRmap.map

123	MV	0
45.5	124	0.419
56.3	32	0.016

3.

```
map2asc -m m -a PCRmap.map AscFile3.txt
```

AscFile3.txt

```
NCOLS 3
NROWS 3
XLLCORNER 10.000000
YLLCORNER 22.000000
CELLSIZE 4.000000
NODATA_VALUE          m
      123.23          m          0
        45.5        124      0.419
       56.34         32      0.016
```

PCRmap.map

123	MV	0
45.5	124	0.419
56.3	32	0.016

Prev legend | Home | Up | Next map2col

resample

Name

resample -- Cuts one map or joins together several maps by resampling to the cells of the result map.

Synopsis

resample [options] Map1 Map2....PCRmapn Result

Map1-n	boolean, nominal, ordinal, scalar, directional (must have the same data type) spatial	
Result	data type of Map1, Map2,...Mapn	spatial

Options

general options (additional options for specifying the location attributes of Result are given in the description section):

cell value assignment

-m

for each cell on Result, the maximum value of the Map1, Map2,...Mapn cells that cover the Result cell is taken (see also description).

subpixellength

-e

subpixellength: *subpixellength* must be a number equal to or between 0 and 2.5 (default 2.5). This options gives the subpixel length as percentage of the cell length on Result. Subpixels are used for calculating the percentage of a Result cell that is covered by a Map1, Map2,...Mapn cell (see also description). If -e 0 is set, the highest accuracy possible (smallest subpixel length) is taken.

missing value assignment

-p

percentmv: *percentmv* must be a number equal to or between 0 and 100, default is 0. This option specifies the assignment of missing values. For each cell on Result, if the percentage of the cell that is covered by non missing value cells on Map1, Map2,...Mapn is less than *percentmv*, a missing value is assigned to the corresponding cell on Result.

Operation

With the **resample** operator one or more input maps Map1, Map2,...Mapn are pasted in a new map Result. The location attributes will be changed to the location attributes of Result. The way this is done depends on some options: you can specify the location attributes of Result with a clone map and paste in that map, or determine the location attributes of Result on basis of the input expressions (cookie cutter or cell size modifier); these options will be described below.

If several maps Map1, Map2,...Mapn are specified they must have the same data type and projection. The location attributes x_{UL},y_{UL} coordinate, number of rows and columns and cell length may be different. The angles of Map1, Map2,...Mapn may be different only if you use **resample** for pasting in a clone map, else these must be the same. If you specify more than one Map1, Map2,...Mapn, these maps may have any spatial location with respect to each other: they may overlap, may be adjoining or they may be separated in space.

Almost in any case, the separate cells on Map1, Map2,...Mapn will not exactly overlap the separate cells on Result. So the raster data on Map1, Map2,...Mapn must be resampled to the raster of Result. For each cell on Result this is done as follows: for each cell on Map1, Map2,...Mapn which is partly or entirely in the cell on Result the area of the Result cell covered by that cell is calculated. This is done by laying down a fine raster of subpixels over the Result cell (default 40 x 40 subpixels per Result cell), and counting the number of subpixels covered by each Map1, Map2,...Mapn cell. These areas are used for assignment of the Result cell value: if the data type of Map1, Map2,...Mapn is scalar or directional an area weighted average of the Map1, Map2,...Mapn cell values is taken, where the weights are the numbers of subpixels covered by the cells. If the data type is boolean, nominal or ordinal the value is taken of the Map1, Map2,...Mapn cell which covers the largest area in the Result cell. If two or more cells both cover the same largest area, the maximum value of these cells is assigned. The maximum value is chosen in any case, if the option -m is set: the areas covered will be totally ignored.

The subpixel length is specified by the option -e *subpixellength*, with a default length of 2.5 % of the Result cell length, which results in 40 x 40 subpixels per Result cell: smaller subpixels will reduce the error made in the computation of the areas, but the time needed to perform the operation will increase.

As above said, the location attributes of the Result map can be specified in three ways:

with a clone map

specifying clonemap

> --clone *Clonemap*: *Clonemap* is taken as clone. If a global clonemap is set as a global option , the option can be omitted, the global clone map is taken as clone map. If the clone map is not set as a global option or if you want to use a different clone map than the global clone, you must specify the clone map in the command line with the option.

This functionality of **resample** is performed if no other options are used than the general options described at the start of the **resample** text. The clone map must be given as described above. Map1, Map2,...Mapn will be pasted in Result which has the location attributes of the clone map. The clone

map and each map Map1, Map2,...Mapn must have the same projection. The other location attributes may be different.

on basis of Map1, Map2,...Mapn (cookie cutter)

specifying border around map(s)

-b *borderwidth*: The smallest rectangle around cells (including missing value cells) is determined. Result will cover an area of this size plus borders or minus borders around this rectangle, where *borderwidth* is the width of the border. A positive *borderwidth* results in a larger map than the rectangle, a negative value in a smaller map. If -b 0 is specified Result will have (approximately) the size of the rectangle.

-c *borderwidth*: idem, the smallest rectangle around non missing value cells is determined.

map expansion or contraction

-x: if the area covered as defined by -b *borderwidth* or -c *borderwidth* contains a fractional number of rows and columns of cells on Result the number of rows and columns is rounded off upwards: the map is expanded (default).

-a: if the area covered as defined by -b *borderwidth* or -c *borderwidth* contains a fractional number of rows and columns of cells on Result the number of rows and columns is rounded off downwards: the map is contracted.

This functionality of **resample** (cookie cutter) generates a Result with location attributes determined on basis of Map1, Map2,...Mapn. One of the options -b *borderwidth* or -c *borderwidth* must be specified and additionally -x or -a and the general options (described at the start of the **resample** text) may be given (optional).

If more than one input map Map1, Map2,...Mapn is given, these must have the same projection and angle; the remaining location attributes may be different. Result will have the same projection and angle as the input maps; the cell size is taken from the first input map (Map1). The x_{UL},y_{UL} coordinates and the number of rows and columns are calculated as follows: first the operations related to the options -b *bordersize* or -c *bordersize* are performed: the smallest rectangle around the edges of the input maps is determined, including or excluding missing values. The rectangle is enlarged or reduced by adding or removing a border at all sides of the map. This new rectangle is the approximate size of the Result, its top left vertex is the x_{UL},y_{UL} coordinate of Result. Rows and columns of cells are laid down in the rectangle, starting at x_{UL}, y_{UL}. If the number of columns or rows needed to fill up the rectangle is a fractional number the rectangle is somewhat (always less than one cel length) expanded or contracted at the right and bottom sides until a whole number of rows and columns of cells fits into the rectangle. This number of rows and columns is assigned to Result. Expansion or contraction is specified with -x (default) or -a, respectively.

to modify cell length

celllength

-r *celllength*: *celllength* is the cell length which is assigned to Result

--unittrue or --unitcell

--unittrue: *cellength* in the option -r is real distance (default)

--unitcell: *cellength* in the option -r is distance in unit cell lengths

map expansion or contraction

-x: if the area covered by the smallest rectangle around the input maps contains a fractional number of rows and columns of `Result` cells the number of rows and columns is rounded off upwards: the map is expanded (default).

-a: if the area covered by the smallest rectangle around the input maps contains a fractional number of rows and columns of `Result` cells the number of rows and columns is rounded off upwards: the map is contracted.

This functionality of **resample** is meant for changing the cell size of the first input map. No clone map must be given. The option -r *cellength* must be set, additionally you can specify --unittrue or --unitcell, -x or -a or the general options described at the top of the **resample** text.

It is quite unlikely that you want to specify more than one map, so first the operation with one map is explained. `Result` will have the projection, angle, x_{UL}, y_{UL} coordinate of the input map `Map`1. The cell length of the input map is changed according to the option -r *cellength* and this length is assigned to `Result`. The area covered by the input map is filled up with cells of the new cell size, starting at x_{UL},y_{UL}. If this results in a fractional number of rows and columns the map is somewhat (less than one new cell length) expanded (default) or contracted until a whole number of columns and rows is reached. This number of rows and collumns is assigned to `Result`.

If more than one input map is given the operation performed corresponds with the operation as a cookie cutter (described above), but you can *not* use the options -b and -c: no borders can be specified. `Result` will approximately have the size of the smallest rectangle around cells (including missing value cells) on the input maps, x_{UL},y_{UL} will be the top left vertex of the rectangle.

See Also

Import map types

---

timeplot

Name

timeplot -- Plots timeseries in a x,y lineplot on the computer screen.

Synopsis

timeplot [options] timeseries1 timeseries2.....timeseriesn

timeseries1-n time series file (s)

Operation

Generates a x,y line plot of the timeseries timeseries1, timeseries2,...timeseriesn. These files must contain the same number of timesteps (i.e. the same number of lines). All data in the timeseries are plotted in one graph. The time column of the timeseries files is printed on the x axis, the data columns on the y axis, using the same scale for all data columns.

The contents and format of a timeseries file must match the dynamic model for which the time series is used.

See Also

Examples

Plotting one timeseries file:

timeplot rain.tss

Index

ADAM module, ADAM module: error propagation

angle, Location attributes, missing values , Creation of a PCRaster map; data import

binding section

- in a Cartographic Modelling script, Script files with binding section

Cartographic Modelling

- example script, Simple script files
- module for, introduction, GIS and Cartographic Modelling

cell

- concept of, Introduction
- length, Location attributes, missing values
- representation, Introduction
- representation, global options, Overview of global options

classified data, data types for, Concepts, kinds of data used in the database

clone map, Creation of a PCRaster map; data import, Overview of global options

clone, global option, Overview of global options

column

- file in simplified Geo-EAS format, Format
- number of, Location attributes, missing values
- table, Format

columntable, global option, Overview of global options

continuous data, data types for, Concepts, kinds of data used in the database

conversion

- between data types, Cutting or joining maps; changing geographical location attributes or data types
- from map to ascii file without x,y, Data export from a PCRaster map
- from map to point data column file, Data export from a PCRaster map

coorcentre, global option, Overview of global options

cutting a map, Cutting or joining maps; changing geographical location attributes or data types

cycle in an ldd, Ldd data type
data type
    assignment to numbers in operations, Command syntax
    concept of, Concepts, kinds of data used in the database, Introduction
    domain defined by, Introduction
    in operations, Command syntax
    subtype, Introduction
degrees
    global option, Overview of global options
diagonal, global option, Overview of global options
digitizing, GIS and Cartographic Modelling
directional data type
    global options related to, Overview of global options
double real cell representation, Introduction
expression, Command syntax
friction paths
    operations with, introduction, Introduction
general systems approach, Operations for transport of material over a ldd
GIS
    module for, introduction, GIS and Cartographic Modelling
initial section
    in a Cartographic Modelling script, Script files with binding section
INT4, Introduction
key
    column, Format
    concept, Concepts, kinds of data used in the database
    field, Format
large, global option, Overview of global options
ldd data type
    global options related to, Overview of global options
    sound/unsound ldd, Ldd data type
local drain direction network, Ldd data type
location attributes
    choice of, Creation of a PCRaster map; data import

concept of, Concepts, kinds of data used in the database
description of, Location attributes, missing values
global options related to, Overview of global options
upper left corner, Location attributes, missing values
xUL, Location attributes, missing values
yUL, Location attributes, missing values
Map Algebra
description of, GIS and Cartographic Modelling, Introduction
map, PCRaster -
concept of, Concepts, kinds of data used in the database
format of, Introduction
material map, Operations for transport of material over a ldd
matrix table, Format
messages printed during execution of operation, Overview of global options
missing value
in maps, explained, Location attributes, missing values
operations
concept (Cartographic Modelling), General approach to Cartographic Modelling
concept (Map Algebra), General approach to Cartographic Modelling
nested pcrcalc operation, Command syntax
operations with local drain direction maps
transport of material, introduction, Introduction
outlet point, Ldd data type
PCRaster
concept of, Introduction
hardware required for, Hardware and software requirements, installation
installation of, Hardware and software requirements, installation
modules of, Introduction
pit, Ldd data type
plain column file, Format
point data column files
concept of, Concepts, kinds of data used in the database
conversion from a map, Creating point data column files, conversion to/from PCRaster maps
conversion to a map, Creation of a PCRaster map; data import

polymorphic behaviour, Introduction
precision of cell values, Introduction
projection, Location attributes, missing values
quotes, use of
    in Cartographic, Dynamic Modelling and GIS operations, Command syntax
REAL4, Introduction
report in Cartographic Modelling, Simple script files
scanning, GIS and Cartographic Modelling
script
    execution of, Simple script files
section keyword, Introduction, layout of the script
small integer cell representation, Introduction
statement, Introduction, layout of the script
table
    concept of, Concepts, kinds of data used in the database
time series
    concept of, Concepts, kinds of data used in the database
transport condition map, Operations for transport of material over a ldd
tuple, Format
unsound ldd, Ldd data type
value field, Format
visibility analysis
    description, Introduction
window operations
    description, Introduction

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

Some updates and additions

- conditional iteration with repeat until
- a note on boolean data and if then else
- inverse distance interpolation
- line numbers in error messages
- arrays of parameters
- selective reports for map stacks in dynamic models
- new functions in pcrcalc
- debugging pcrcalc scripts
- argument substitution in pcrcalc
- putting a global option IN a pcrcalc script
- map2col: new feature of -f option
- resample: new option -k
- list of current unsolved bugs
- printing PCRaster maps

Clickpath: Index Versions: page: 20 November 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

Conditional iteration with repeat until

Conditional iteration (looping) allows to perform model code a conditional number of times. The syntax is:

repeat { Statements } **until** BooleanCondition ; The Statements are executed repeatedly until the BooleanCondition is true. For example:

```
nrLoopsExecuted=0;
repeat {
 nrLoopsExecuted=nrLoopsExecuted+1;
} until nrLoopsExecuted eq 4;
```

The statement `nrLoopsExecuted=nrLoopsExecuted+1` is executed 4 times.

If the BooleanCondition is spatial (it evaluates to a map) then the Statements are executed until no cells in BooleanCondition are false(0). In other words execution of the loop is stopped if all cells are true(1) or missing value. Note that the execution is not stopped at a per cell base, but when ALL cells meet the specified condition. In theory, stopping at a per cell base might decrease execution time, but the additonal bookkeeping of which cells needs one iteration more at each loop will neutralise this gain.

A more realistic example is an iteration to fit a certain function, with a series of succesive estimates that will stop if the next estimate only differs a certain small epsilon value.

```
prevEst= ...initial estimate ... ;
repeat {
  nextEst= ... estimate ... ;
  difference = nextEst-prevEst;
  prevEst=nextExt;
} until difference < 0.00001;
```

If the result of nextEst is a map, then this loop will continue until each cell in difference has a value smaller than 0.000001.

Current bug and possible work around

In the current version of the repeat-until construction there is a serious flaw: under certains conditions map data that is referenced within in the repeat code block for the last time in the script is prematurely released. This bug will terminate the model run with a message like: "RUN TIME ERROR prevEst has no value." The solution is to define a dummy operation on prevEst just after the repeat until construct:

```
prevEst= ...initial estimate ... ;
repeat {
  nextEst= ... estimate ... ;
  difference = nextEst-prevEst;
  prevEst=nextExt;
} until difference < 0.00001;
# dummy operation
prevEst = prevEst + 0;
```

Clickpath: Conditional iteration with repeat until Versions: page: 24 November 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

A note on boolean data and if then else

It seems that the use of boolean true/false data in models is a difficult subject for many users. I see a lot of very complicated expressions in user models that are unnecessary and error prune. This week a small bug was discovered in pcrcalc, that only happens when users write this kind of very complicated expressions. The bug will NOT yield wrong results but simply refuses to run certain models.

Very often I see model fragments like:

```
nearGroundWater = if ( GroundWaterLevel lt 20 then 1 else 0);
```

or

```
nearGroundWater = boolean(if ( GroundWaterLevel lt 20 then 1 else 0));
```

This is identical to the much shorter and faster:

```
nearGroundWater = GroundWaterLevel lt 20;
```

the first expression of the if-then-else is already a boolean expression consisting of 0 (false) and 1 (true) and possible missing values, so there is no need to write the values of 1 and 0 explicit. This happens often when using any of the boolean and comparison operators.

More dangerous is the common believe that a statement like:

```
nearGroundWater = if ( GroundWaterLevel lt 20 then 1); # WRONG!
```

will result in a true/false map, as in the statements given above. This is wrong: if without the else will create missing values not 0 (false).

People also tend to think in terms of "if this then that happens, if not then if this the case then bla bla...". This results in statements like:

```
A      = aDepth lt 20;
B      = bDepth lt 20;
result = boolean(if( A eq 1, 1, if(B eq 1, 1, 0)));
```

the same is better expressed as:

```
result = aDepth lt 20 or bDepth lt 20;
```

to summarize in a few guidelines:

1. if-then and if-then-else are two totally different operations:
   - if-then is only to create missing values in maps.
   - if-then-else is to make a selection between 2 values.
2. Use the syntax: `if( then else )` and `if( then )` instead of `if( , , )` and `if( , )` to see better which type of operation you use.
3. Avoid patterns like `if(` *`expression`* `then 0 else 1)` or `if(` *`expression`* `then 1 else 0)` in your model. In most cases these patterns can be rewritten. If these patterns are nested like in `if (.. (if (...` try rephrasing in terms of `and, or, xor` and `not`.

Clickpath: Boolean data Versions: page: 18 November 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

Inverse distance

```
pcrcalc result = inversedistance( mask, points, idp,radius,maxNr);

synopsis
 result  scalar, spatial
 mask    boolean, spatial or non-spatial
 points  scalar, spatial
 idp     scalar, spatial or non-spatial
 radius  scalar, spatial or non-spatial
 maxNr   scalar, spatial or non-spatial
```

Options

--unittrue,--unitcell

> --unittrue radius is measured in true distance (default). --unitcell radius is measured in number of cell lengths.

Description

Interpolate values from point data.

Operation

for each cell, having a value 1 (TRUE) in *mask*, a value is interpolated using weighted average of all non missing value points on *points*. The weighted average method is an inverse distance scheme with a weight of d^{-idp}.

radius allows to select only the points at a distance less or equal to the cell a value is computed. If *radius* is 0 or less then there is no restriction on the distance towards the cell; all points are candidate to contribute to the computation.

maxNr allows to select only the *maxNr* closest points as the maximum number of points used in the computation. Less points are used if if no *maxNr* points can be found with in the *radius*. If *maxNr* is 0 or

less then all points are candidate to contribute to the computation.

A missing value is generated if the the combination of *radius* and *maxNr* setting yield no points at all.

Inversedistance is typically used to interpolate sparse point samples to a continuous surface of values. An *idp* value of 2 is commonly used to achieve a smooth surface. A result with *idp* value of 2 is also much faster to compute then another *idp* value.

A common use of inversedistance is in combination with timeinput, to create a value surface each timestep. This setup will also handle the case where timeseries have missing values (1e31): in that case a point is not defined in that timestep and will not take part in the computation of the weighted average.

Notes

- The execution time of this function increases linear with the number of non-missing values on *points*
- Using all points in the interpolation by setting both *radius* and *maxNr* to 0 will gives the fasted execution time.

Example 1

```
binding
 # a map with columns id's MV's elsewhere
 pointId = pointid.map;
 # a map with 1 (TRUE) for area of interest
 mask    = mask.map;
 # a timeseries possible with missing data (1e31)
 inputTss= input.tss;
....
dynamic
 points = timeinputscalar(inputTss, pointId);
 # use all points if radius is 0 and maxNr is 0
 result = inversedistance(mask, points, 2,0,0);
```

Example 2

Since radius can be spatial we may specify a different radius for each cell. In combination with spread() we can enlarge the radius in such a way that at least 1 point is always selected in areas where no sampling points are near. Note that we add one celllength to adjust for cell roundings.

```
binding
 # a map with columns id's MV's elsewhere
 pointId = pointid.map;
```

```
 # a map with 1 (TRUE) for area of interest
 mask    = mask.map;
 # a timeseries possible with missing data (1e31)
 inputTss= input.tss;
....
dynamic
 points = timeinputscalar(inputTss, pointId);
 spreadSurface = cover(points,0);
 distanceToNearestPoint = spread(spreadSurface,0,1)+celllength();
 # 5000 is the default radius
 radius = max(spreadSurface,5000);
 # use at max 5 points within radius
 result = inversedistance(mask, points, 2,radius,5);
```

Clickpath: Inverse distance Versions: page: 19 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

Line numbers in error messages

All error message of pcrcalc have the format: *`scriptFileName`*:*lineNr*:*colNr*:ERROR: A descriptive message

The message is a list of positions separated by the symbol :

`scriptFileName`

Name of the script file or the ?-symbol if a statement is issued from the command line.

`lineNr`

The line number within the script file.

`colNr`

The column number where the error is detected.

In most cases, pcrcalc will point to the exact position where the error is present. There are a few occasions that the actual (syntax) error is before the position that pcrcalc reports. One of these occasions is the generic syntax error like:

```
example.mod:5:36:ERROR: Syntax error at symbol ';'
```

In this example, the symbol ";" at line 5, character position 36 is not expected. This may be caused by earlier symbols in the script.

Clickpath: Line numbers Versions: page: 19 November 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

Arrayed variables in PCRcalc

Document version 4.0 By Willem van Deursen and Cees Wesseling, PCRaster Environmental Software VOF

DISCLAIMER: The new features documented here are considered to be in alpha status. Things may change a bit in later releases. This release is intended to get feedback from users who want to apply these features. As always, we do welcome ideas and suggestions but for alpha software we can not garantuee the same level of support as for our regular software.

Introduction

PCRcalc allows for the construction of arrayed variables. The concept reduces the need for copying parts of the script several times to create similar behavior for a number of species or related components.

The clearest need for this type of concept comes from simulating ecosystem dynamics, where the main goal is to model the interaction between species. Although ecosystem dynamics deals with a number of species, these species demonstrate similar (not identical) behavior. Until the introduction of arrayed variables, this type of models was constructed in PCRaster by copying the section that described the first species and substitute the parameter values for the second species. This approach yields a script that can become very long and is becoming more and more difficult to maintain. Changes made to the process descriptions have to be implemented in several similar sections, and parameter values are scattered throughout the script. The use of arrayed variables improves this situation because it allows for a structured way of describing the similar processes in the different species, and it allows for the species to be initialized from external files.

Identification and access to arrayed variables

The introduction of the arrayed variables is straightforward. In the binding section of the script we need to identify the arrayed variable, while in the initial and dynamic section we need to use the **foreach** construct to iterate over the array. Furthermore, we developed index... functions to create an interface to external parameter files. The identification of the arrayed variables is in the binding section.

```
binding
  PlantSpecies = [
        Species1,
        Species2,
```

```
      Species3];
 HerbivoreSpecies = [
      Herbivore1,
      Herbivore2];
```

This introduces one **array** index type, called PlantSpecies, in the script. PlantSpecies (the type) has three names of type **array-index**: Species1, Species2 and Species3. Additionally, the type HerbivoreSpecies is introduced. This type is composed of two indices: Herbivore1 and Herbivore2. Within the initial and dynamic part of the script we can now create and access arrayed variables using these types:

```
# this is the initial or dynamic section
Variable1[Species1] = ...;
Variable1[Species2] = ...;
Variable2[Herbivore2] = ...;
# this is the initial or dynamic section
foreach species in PlantSpecies {
  Variable1[species] = ...;
  Variable2[species] = ...;
}
```

Or, nested

```
# this is the initial or dynamic section
foreach species in PlantSpecies {
  foreach herbivore in HerbivoreSpecies {
   VarInteraction[species][herbivore] = ...;
  }
}
```

Using external parameter files for arrayed variables

Using index... functions we can initialize arrayed variables (parameter-) values from parameter definition files. There are three types of index functions:

- **index** initialize variable with a set of input maps.
- **indextable** initialize variable with a set of lookuptables for use in lookup functions.
- **index*type*** initialize non-spatial variable with a set of numbers

Similar to the lookup... functions, the index*type* functions is actually a family of functions with the members indexscalar, indexboolean, indexordinal and indexnominal. Since the index... functions are actually mapping external files on modelnames, the index... functions can only be used in the binding section. Although this may seem like a large limitation on the use of the index... functions, in practice this only means that the parameter values read from the parameter files can only be used on the right side of the assign-token (=-token), which makes sense. The syntax of index... is:

VarName[TypeName] = *indexfunction*(TableName); TableName is the name of an external column file which stores the parameter values. TypeName is the name of the type used for index definitions

VarName is the name of the parameter as it appears in the parameterfile. This name can be overruled using the binding operations for linking external names to internal names (see below). Thus, supposing Species.tbl contains some parameters Par1 and Par2 for the array type Species (members Species1, Species2 and Species3) we can access this value using

```
binding
 Species = [Species1, Species2, Species3];
 Par1[Species] = indexscalar(Species.tbl);
 Par2[Species] = index(init_a.tbl);

dynamic
  foreach s in Species {
    myvalue[s] = Par2[s] * Par1[s] + ...;
  }
```

In this case, the parameterfile Species.tbl should look something like:

```
Par1   Species1  0.03
OtherParameter Species1  ...
Par1   Species2  0.05
OtherParameter Species2  ...
Par1   Species3  0.02
OtherParameter Species3  ...
Par2   Species1  par2_sp1.map
Par2   Species2  par2_sp2.map
Par2   Species3  par2_sp3.map
```

Note that the parameter name as it appears in the binding section must exactly match the name as it appears in the parameter file. And the rows in the parameterfile do not have to be in a certain order, except if the wildcard * (NOT IMPLEMENTED) is used, as shown in the next example. This functionality is also available for more dimensions: suppose we have two types A and B, and we have a parameter file Inter.tbl which describes the interaction between these types. Parts of a sample script might look like:

```
binding
  A = [a1,a2,a3];
  B = [b1,b2,b3];
  Interaction[A][B] = indexscalar(Inter.tbl);
 dynamic
  foreach a in A {
   foreach b in B {
    ...
    VarSomething[a][b] = ... + Interaction[a][b]* ...;
    ...
    }
}
```

with the parameterfile Inter.tbl looking like

```
Interaction  a1 b1 0.2
```

```
Interaction  a1 b2 0.35
Interaction  a1 b3 0.33
Interaction  a3 b1 0.35
Interaction  a2 b3 0.4
Interaction  * * 0.10 # Default (NOT IMPLEMENTED)
```

Note 1: The order of columns in the parameter file define the order in which the types appear in the indexscalar-statement. The first column defines the name of the parameter, the middle columns are in the order of array types used on the parameter, and the last column is the result value of the index... operation.

Note 2: **NOT YET IMPLEMENTED** Not all possible combinations have to be defined; only those actually used in the script do have to have a value. It is also possible to define default values (see the last line in the example), in which more specific definitions always overrule more general definitions, and the first occurrence overrules later occurrences (see also lookup-table rules).

Linking variable names to external names

The binding section allows for linking variable names to external names. Within the script we might want to use different names for variables than the names used in the database. Also for the arrayed variables, the binding section allows for linking model names to script names.

```
binding
ArrayVariable1 = [
  Modelname1 = Externalname1,
  Modelname2 = Externalname2,
  Modelname3,
  ...
  ];

binding
   PlantSpecies = [
  Que = Quercus,
  Fag = Fagus
  ];
```

This example links the modelnames Que and Fag to the external names Quercus and Fagus. In this case, the names which are used in the parameterfiles are the external names Quercus and Fagus, while the names used in the script are the modelnames Que and Fag. Using the above example, the parameter file specifies the external names and looks like

```
Quercus ImportantValue1 0.03
Fagus ImportantValue1 3.56
```

The script uses the modelnames and looks like

```
binding
 ImportantValue1 = indexscalar[Species.tbl];
```

The same is true for the parameter name. Assuming the same parameter file:

```
binding
PlantSpecies = [
  Que = Quercus,
  Fag = Fagus
  ];
MyParam = ImportantValue1;
...
MyParam[PlantSpecies] = indexscalar(Species.tbl);
```

Switching arrayed variables off

Consider the case in which we want to build a complex ecosystem simulation model for a region. We identified four species in the region, but we only collected parameter data for two of these species. We want to construct the model for the interaction between all four species, but for the time being we can only run the model for the two measured species. PCRaster allows for indexed variables to be switched on and off, thus defining whether they should be part of the actual simulation or not. The syntax for this switching is to add a - (minus) sign before each index in the type-definition that should not be modeled.

```
binding
PlantSpecies = [
  -Species1 = Quercus,
  Species2 = Fagus,
  -Species3];
HerbivoreSpecies = [
  Herbivore1,
  Herbivore2];

 ImportantValue1[PlantSpecies] = indexscalar(Species.tbl);
```

In this example PlantSpecies Species1 and Species3 are defined but not included in the simulation. The indexscalar statement thus yields only one variable, ImportantValue1[Species2]. This ImportantValue1 for Species2 has to be defined in the table Species.tbl, otherwise an error occurs. For Species1 and Species3 the ImportantValue1 does not have to be defined, since the simulation will never try to access these values. There is one catch when switching off an index. A switched off index can not be explicitly addressed in a statement like in:

```
DerivedValue[Species3] = ImportantValue1[Species3] + ...;
```

But switched off indices are usable in defining **array-set**s and **foreach** structures, as shown in the next sections. Once the datasets for Species1 and Species3 are available, it is now very simple to incorporate them into the model by erasing their - (minus-sign). Note that we can use the + (plus sign) for each index included in the simulation, but this is default behavior.

```
binding
PlantSpecies =  [
  -Species1 = Quercus,
  +Species2 = Fagus,
```

```
  -Species3];
HerbivoreSpecies = [
  +Herbivore1,
  +Herbivore2];
```

Sets: combining indices

If one specifies the name of the array type in the foreach construct, the construct will iterate over all indices of that array type, that are not switched off. If one wants to iterate only over a subset of indices, there are two options, defining subsets (named **array-set**) or using the except clause of foreach. Subsets are defined within the array type definition using the construction

```
setName = ( indexName1, indexName2 );
```

For example:

```
B = [b1,-b2,b5,b4,b3,
      Set1_3 = ( b1, b2, b3 ),
      Set1_4 =  ( Set1_3, b4 )
     -Set45=  ( b4, b5 )
    ];
```

The above example shows a number of features:

- Sets only include indices that are active. Set1_3 will only contain b1 and b3 since b2 is switched off.
- Sets can be used within set definitions. Set1_4 will contain b1,b3 and b4.
- Sets can be switched off.

Arrays and foreach: A formal definition

In the previous sections, all new concepts are introduced. We now try to formalize these concepts. The syntax of **foreach** is: foreach *iterator* in *indexCollection* { *Block-Of-Statements* } **or** foreach *iterator* in *indexCollection* except *indexCollection*{ *Block-Of-Statements* }

In this syntax *iterator* is a variable name that can only be used in the *Block-Of-Statements*. The *indexCollection* is either a single variable name or a list of variable names: *indexVariableName* **or** (*indexVariableName, indexVariableName, ...*)

An *indexVariableName* can be any of the following variables:

- An **array** name
- An **array-index** name
- An **array-set** name
- The name of an *iterator* of a possible foreach construct enclosing this foreach construct.

The following examples summarizes these definitions:

```
B = [b1,-b2,b5,b4,b3,
```

```
    Set1_3 = ( b1, b2, b3 ),
    Set1_4 = ( Set1_3, b4 )
   -Set45 = ( b4, b5 )
   ];
foreach  h in B {
 foreach i in Set1_4 except (h,b1) {
 ...
 }
}
```

- The **array** variable is B
- The **array-index** variables are b1,b2,b3,b4,b5
- The **array-set** variables are Set1_3,Set1_4,Set45
- The iterators are h and i, h is in the second foreach the iterator of an enclosing foreach.

The *Block-Of-Statements* is executed a number of times with the *iterators* holding a distinct array-index variable. The possible values of the iterators is computed by a result set of the *indexCollection* of the in-clause with all values mentioned in the *indexCollection* of the optional except-clause removed. Switched off array-indices or array-sets are simply discarded in the compution of the result set. The execution order is always in the order of definition of the array-indices (here b1,b2,b5,b4,b3). Implicit duplication of array indices has no effect, the foreach construct is only evaluated once for each distinct array index variable. So:

```
foreach  h in (B,Set1_4,b4) { ...
```

Is valid and will only result in 4 times evaluation of foreach in the following order: b1,b5,b4,b3.

Reporting output for arrayed variables

Reporting arrayed variables are done under the same rules as normal variables, concerning external names that are optionally defined in the binding. To discern between the different array index names, each array item is written under a name containing all array index names, with a dash '-' in between.

Clickpath: Arrays Versions: page: 19 November 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

Selective reports for map stacks in dynamic models

Pcrcalc version of Feb 28 1998 or later allow to define at which timesteps a map must be written to the permanent database (in other words your harddisk). This feature consists of the following extensions to the modelling language:

- Defining a sequence of report moments in the timer section of the script by a symbolic name *reportName*.
- Using that symbolic name in the report statement in the dynamic section: **report(***reportName***)**.
- Definining a sequence of report moments in the report statement in the dynamic section: **report(***reportMoments***)**.
- Introduction of the keyword **endtime** as the symbolic name for the last timestep.
- Introduction of the keyword **reportdefault** as the symbolic name for the reports without an explicit definition of the report moments.
- **NOTE** Selective reports only apply for map stacks, when writing to a timeseries file they are ignored, a timeseries file will always contains data for **all** timesteps.

To explain this feature, we use the following minimal script

```
timer 1 1000 1;
initial
dynamic
 report stack1_ = input.map;
 report stack2_ = input.map;
```

The script simply copies the input.map to 2000 maps named stack1_0.001, stack2_0.002 up to stack1_1.000 and stack2_0.001 up to stack2_1.000.

Now suppose we want to write to stack stack1_ only at step 1, 10, 900, 1000 and stack2_ at every fifth step. We then define and use two report moment definitions in the timer sections and apply them to the reports:

```
timer 1 1000 1;
 rep1 = 1,10,900,endtime;
```

```
 rep2 = 5+5..endtime;
initial
dynamic
 report(rep1) stack1_ = input.map;
 report(rep2) stack2_ = input.map;
```

Different moments in a report moment definition are seperated by ',' as in 1,10,900,endtime that defines 4 moments. A range of moments can be given by the syntax *start*+*step*..*end*. *Step* increases the moments up and including *end*. In the example above stack2_ will be reported at timestep 5,10,15,25, etc. until 1000. If the + is omitted a step value of 1 is assumed. In other words 5..10 and 5+1..10 will both result in 5,6,7,8,9,10.

A report moment definition can also be placed within the report statements:

```
timer 1 1000 1;
initial
dynamic
 report(endtime)   stack1_ = input.map;
 report(1,5+3..12) stack2_ = input.map;
```

In the example above stack1_ is only reported at the last step and stack2_ at 1,5,8,11.

A special report definition is **reportdefault**. Defining this one in the timer section causes all stack reports without explicit moments to report only at particular moments:

```
timer 1 1000 1;
 reportdefault = 900+5..endtime;
initial
dynamic
 report stack1_ = input.map;
 report stack2_ = input.map;
```

Both stack1_ and stack2_ are reported at steps 900,905,910..1000.

Clickpath: Report Versions: page: 19 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

New functions in pcrcalc

streamorder

Synopsis

```
pcrcalc result = streamorder(ldd)
ldd    is ldd
result is ordinal
```

Strahler order stream network. The classification of stream networks was originally developed by Horton, and modified by Strahler. Following the scheme of Strahler, streamorder designates an order 1 to the smallest channels, which are the cells with no upstream cells connected to that cell. Where two channels of order 1 join, a channel of order 2 results downstream. In general, where two channels of order i join, a channel of order i+1 results. **NOTE** The september 1997 print of the manual has this command, but incorrectly documents the return type as nominal.

catchmenttotal

Synopsis

```
pcrcalc result = catchmenttotal(ldd,amount)
ldd    is ldd
amount is scalar
result is scalar
```

Calculate total for the entire upstream area. Note that this function is identical to accuflux, except that accuflux does not accept negative values.

fac

Synopsis

```
pcrcalc result = fac(expr)
expr is scalar
```

```
result is scalar
```

Computes the faculty or factorial of a natural positive number. Domain errors result in a MV. Overflow occurs on 35 and bigger, resulting in a MV.

spreadmax, spreadmaxzone (EXPERIMENTAL)

Identical to spread and spreadzone but with a fourth parameter, a maximum spread distance. Areas that are not reached are given the value MV for spreadmax and 0 for spreadmaxzone. On the final names of these commands is not yet decided thus the EXPERIMENTAL status.

kinematic

Synopsis

```
pcrcalc newQ = kinematic(ldd,Qold,q,alpha,beta,deltaT,deltaX)
ldd  is ldd
all other inputs are scalar:
Qold in m3/s
deltaT s/timestep
deltaX cellsize
newQ in m3/s
```

Printed manual page available on request Notes on features and unresolved bugs here

Clickpath: New functions Versions: page: 19 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

kinematic, unresolved bugs and features

Clickpath: Kinematic Versions: page: 19 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences Utrecht University
The Netherlands

Universiteit Utrecht

debugging pcrcalc scripts

A frequent problem when writing scripts is that sometimes the results have unwanted missing values due to invalid ranges on some functions, such as division by zero or taking the square root of a negative value. The experimental -d option of pcrcalc can help in finding the places where this occurs. But only if you follow the **convention** that at each creation point of a map, the map must have non missing values at each location where the boolean areamap (or clone map) has a true (1) value. For example, the script test.mod:

```
areamap mask.map;
timer 1 100 1;
dynamic
 Result.map = (VarA.map*VarB.map)/SomeZero.map;
```

If the script is called like `pcrcalc -d debug.map -f test.mod` then at each timestep when Result.map receives a new map computed, the new map is checked if it has any missing values where the areamap (mask.map) is true (1). If so, pcrcalc terminates immediately with an error message, and writes the map debug.map. The map debug.map may contains the following values: MV: where a missing value is in the areamap 0 : where a 0 value is in the areamap 1 : where a 1 value is in the areamap and the checked does not have a missing value **2** : where a 1 value is in the areamap and the checked does have a missing value

With this technique you find may find the points where additional work is needed. In the example above, we know that the formula only applies to cases where SomeZero.map is larger than 0 and other cases should have the value 0:

```
Result.map = if (SomeZero.map gt 0 then (VarA.map*VarB.map)/SomeZero.map else 0);
```

Another solution is to cover the generated missing values:

```
Result.map = cover((VarA.map*VarB.map)/SomeZero.map, 0);
```

This kind of debugging is of limited use if one applies operations that generate missing values on purpose (such as if then without the else clause) in this way:

```
Temp.map = if(VarA.map eq VarB.map then VarC.map);
Result.map = cover(Temp.map, 0);
```

If the -d option is used then the creation of Temp.map will create an error message. Rewriting the operation fixes this:

```
Temp.map = if(VarA.map eq VarB.map then VarC.map else 0);
```

Clickpath: Debugging Versions: page: 19 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

substitution of arguments with a $-construct

In both command line expressions and model scripts, parts of the model can be substituted from shell-like arguments.

```
For example, sum_2.mod :
$1 = $2 + $3;
Calling
pcrcalc -f sum_2.mod sum_2.map add_1.map  add_2.map
Is equal to
pcrcalc "sum_2.map = add_1.map + add_2.map;
On the command line, the model and the arguments need to be separated by ;; .
For example:
pcrcalc "$1 = $2 + $3 ;;" sum_2.map add_1.map add_2.map
```

Basic substitution rules:

- everything may substituted, even operators, functions, etc.
- if a substitution fails, blanks are returned: no error message is printed, the resulting string might be incorrect. (use -t to check substitution without running the model)
- A $-sign followed by a number refers to an argument on the command line, e.g. $1 $2
- If the $-sign is followed by a non-numerical string the argument is a shell or environment variable, for example $RESULT.
- Simple arguments may be enclosed in curly braces, e.g. `$1 = $2 + $3` equals `${1} = $2 + $3;` This enables prefix-substitution of a variable such as `${1}.map = $2 + $3;`

Advanced substitution rules:

- A range of parameters can be given in the ${from,to} construct. For example: `$1 = max(${2,3}) ;; max.map in1.map in2.map` becomes `max.map = max(in1.map,in2.map)`
- n in the first or second arguments denotes the number of arguments: `$1 = max(${2,n}) ;; max.map in1.map in2.map in3.map` becomes `max.map = max(in1.map,in2.map,in3.map)`
- The ${from,to} construct prints an ',' between every argument. Another argument separator can be given explicitly: `$1 = ${2,n,+} ;; sum.map in1.map in2.map in3.map` becomes `sum.map = in1.map+in2.map+in3.map`
- A wrapper around each arguments can be given: `$1 = ${2,n,+,sqrt($)} ;; sumsqrt.map in1.map in2.map in3.map` becomes `sumsqrt.map = sqrt(in1.map)+sqrt(in2.map)+sqrt(in3.map)`
- In the 4th argument, the wrapper, a $ is given for where the argument should be inserted. That $-sign is optional, allowing the following construct: `$1 = (${2,n,+}) / (${2,n,+,1});; av.map`

`in1.map in2.map in3.map` becomes `av.map = (in1.map+in2.map+in3.map) / (1+1+1)`

- white space in argument 3 and 4 is kept in the substitution: `$1 = ${2,n,and,not $} ;; andnot.map in1.map in2.map in3.map` becomes `andnot.map = not in1.mapandnot in2.mapandnot in3.map` while `$1 = ${2,n, and ,not $} ;; andnot.map in1.map in2.map in3.map` becomes `andnot.map = not in1.map and not in2.map and not in3.map`

Additional notes:

- Currently, comments (#) are not allowed within a substitution
- Note that ${n} denotes the last argument, while $n means an environment variable named n.
- Note that is an extra substitution level, so one needs to give $$1 in an UNIX shell, if the command line expression is enclosed in "-symbols. This is not neccessary if it is enclosed in '-symbols.
- If you are trying $-constructs and you get the message: ERROR: parse error near line '?' near symbol '?' Then two errors are frequent:
  - you forgot the delimit the expression and the arguments by ;;
  - the substitution went wrong. First try -t, to see what the result of the substitution is. As said before, the substitution mechanism silently ignores most errors, and returns blanks. So be warned.

Clickpath: Argument substitution Versions: page: 19 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

global options can be given in the first line of a model script.

Global options that aply to a script can be stored in the script itself instead of typing it on the command line when the script is called. The first two characters of the first line of the script should read #!. After #! the global options can be specified. For example

```
#! --lddfill --radians
..etc.
```

Note that the #! should exactly be the first positions on the first line. Leading (empty) lines or spaces will change the line to a normal comment, and no global options from that line are set. No error messages are printed in such a case.

The remainder of this discussion on global options in the script applies only to those who use pcrcalc from a UNIX-like shell. The syntax of this first line is modelled after the alternative interpreter syntax of UNIX shells. If the first 2 characters of this first line are #! then all words on that line that can be recognized as global options, are parsed as global options, overriding global options at the command line. For example, example.mod:

```
#! /usr/local/bin/pcrcalc -F --lddfill
..etc.
```

To be fully compatible with the UNIX alternative interpreter notation, one needs to specify the -F flag on the first line if and only if one wants to put global options on the first line. Note that pcrcalc will always scan the first line of a script, even if the script is not executed by itself, in other words: pcrcalc -f example.mod is equal to chmod +x example.mod example.mod In both cases the --lddfill option will be activated. Since pcrcalc scans the first line itself, these feature is also enabled on NON-UNIX OS's (MSDOS).

-F is only neccessary and can only be used if one wants to specify global options. If the goal is to only describe pcrcalc as the interpreter of the script then use -f. This is due to a limitation in the syntax of an alternative interpreter, only one option can be specified. For example, example2.mod:

```
#! /usr/local/bin/pcrcalc -f
..etc.
```

Clickpath: Options Versions: page: 19 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

update map2col -f option:

The -f option of map2col(dated Feb 28 1998 and up) accepts multiple format definitions. For example: `map2col -f"10.3f 10.3f 4.0f" input1.map input2.map result.col` Will print the first and second column in 10.3f format and the third and remaining columns in 4.0f format. Note that the format string must be enclosed in double quotes (") to pass the string as a single argument to the command, if more than one format definition is given.

If the output has more columns than there are format definitions, the last definition is used for the columns with no format definition. Thus, giving one format definition will result in output with identical formatting for all columns.

In append mode (-a), the format definitions only apply to the appended columns; the first format definition is applied to the first APPENDED column not the first output column.

Clickpath: Map2col Versions: page: 19 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

resample -k option:

The -k option of resample (dated Feb 28 1998 and up) sets the minimum and maximum of the result map to the minimum and maximum of the input maps.

Clickpath: Resample Versions: page: 19 August 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

Known Bugs in pcrcalc

- tan(180) is not 0 The tangent of 180 degrees a very small number instead of 0 (8.7e-8). The workaround is to add an if statement:

```
pcrcalc test.map = if ( v == 180 then 0 else tan(v))
```

- lookup functions misbehave if directional data of exactly 360 degrees is used Consider the operation

  `pcrcalc test.map = lookupboolean(asp.tbl,aspect.map)`

  with aspect.map being directional and asp.tbl with the contents:

```
[0,135>   0
[135,225> 1
[225,360] 0
```

  The range is [225,360] will produce missing values, which is wrong. The workaround is to make the range open:

```
[0,135>   0
[135,225> 1
[225,   ] 0
```

Clickpath: Bugs Versions: page: 20 November 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k

Department of Physical Geography Faculty of Geosciences
Utrecht University The Netherlands

Universiteit Utrecht

Harcopy output: printing PCRaster maps

There are two ways to print PCRaster maps. The first option is mostly used, the second option is a bit more difficult and does not always result in better prints.

Dumps of PCRaster maps visualised with display

This is done by displaying the map on the screen, and making a dump of the screen. You will get a jpg, bmp or gif format, depending on the screen capture programme. You need a screen capture programme, for instance Paintshop running in Microsoft Windows (evaluation edition can be downloaded from http://www.jasc.com/, free of charge), or capture express (http:/www.captureexpress.com, free of charge).

1. Display the map.
2. Capture the map display from the screen. In Paintshop, use Capture->Setupto define the options and then Capture->Start to grab the map.
3. Edit (if needed) and save the captured map in your screen capture programme or copy-paste it to e.g. powerpoint.

Converting PCRaster maps to postscript

Direct conversion from PCRaster map format to encapsulated postscript format. Resulting file can be printed on a postscript printer, or imported to a graphics programme (eg Freehand 3, Freehand 7 does not work).

Clickpath: Printing Versions: page: 19 November 2004, website: 18 February 2005. Webmaster: ln.uu.oeg@gnojed.k